NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIFORNIA

THESIS

**SOFTWARE DEVELOPMENT PROCESS FOR THE
AVIATION MISSION PLANNING SYSTEM (AMPS):
A CASE STUDY**

by

Keith R. Edwards

December, 1995

Thesis Advisor: Martin J. McCaffrey

Approved for public release; distribution is unlimited.

	REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE December 1995	3. REPORT TYPE AND DATES COVERED Master's Thesis	
4. TITLE AND SUBTITLE Software Development Process for the Aviation Mission Planning System (AMPS): A Case Study		5. FUNDING NUMBERS	
6. AUTHOR(S) Keith R. Edwards			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey CA 93943-5000		8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSORING/MONITORING AGENCY REPORT NUMBER	

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.	12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*

The DoD software development environment is one in needed transition. Many of the old methodologies have been less than effective for software development. Emerging methods and techniques, for instance, evolutionary development and incremental delivery, and the use of CASE tools, are supported by a new set of flexible standards. MIL-STD-498, Software Development and Documentation, and the coming commercial equivalent, emphasize flexibility, tailoring, and value-added activities. The Aviation Mission Planning Systems (AMPS) software development effort, is a study in the employment of innovative, emerging methods and techniques in this evolving environment. Originally a prototype, the AMPS program will now lead to a production system. The development process for the supporting software is now undergoing a transition. This thesis examines this transition and discusses several process improvement considerations as they relate to the AMPS software development process. Additionally, this thesis explores several areas of concern surrounding the AMPS software development process transition, and suggests possible mitigation approaches.

14. SUBJECT TERMS Software Development, Aviation Mission Planning System (AMPS), Software Prototypes			15. NUMBER OF PAGES 119
			16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18 298-102

SOFTWARE DEVELOPMENT PROCESS FOR THE AVIATION MISSION PLANNING SYSTEM (AMPS): A CASE STUDY

Keith R. Edwards
Captain, United States Army
B.S., University of Delaware, 1984

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN MANAGEMENT

from the
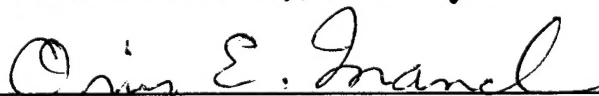
NAVAL POSTGRADUATE SCHOOL
December 1995

Author: _____
Keith R. Edwards

Approved by: _____
Martin J. McCaffrey, Thesis Advisor

_____
Orin E. Marvel, Associate Advisor

_____
Reuben T. Harris, Chairman, Department
of Systems Management

iii

ABSTRACT

The DoD software development environment is one in needed transition. Many of the old methodologies have been less than effective for software development. Emerging methods and techniques, for instance, evolutionary development and incremental delivery, and the use of CASE tools, are supported by a new set of flexible standards. MIL-STD-498, <u>Software Development and Documentation</u>, and the coming commercial equivalent, emphasize flexibility, tailoring, and value-added activities. The Aviation Mission Planning Systems (AMPS) software development effort, is a study in the employment of innovative, emerging methods and techniques in this evolving environment. Originally a prototype, the AMPS program will now lead to a production system. The development process for the supporting software is now undergoing a transition. This thesis examines this transition and discusses several process improvement considerations as they relate to the AMPS software development process. Additionally, this thesis explores several areas of concern surrounding the AMPS software development process transition, and suggests possible mitigation approaches.

TABLE OF CONTENTS

I. INTRODUCTION

This thesis examines past and present methodologies for the procurement of mission critical computer resources (MCCR) for major weapon systems. Additionally, a case study is made of the software development process employed for the Army's Aviation Mission Planning System (AMPS). The methodologies employed in the development of this system software are evolutionary. Originally developed as prototype software, the AMPS development team has been tasked to transition to the development of production software. Their efforts to evolve the software development process from prototype to production orientation are in many ways illustrative of the initiatives necessary in the migration from the inefficiencies of the past, to process optimization of the future.

A. GENERAL

Mission critical computer resources, particularly mission software, have traditionally been developed in an incremental fashion (e.g. a sequence of "builds" with multiple configurations developed concurrently) , or similarly, in a step-by-step process where previous requirements are met before proceeding to the next step in the sequence.

Defense software development activities (as reflected in DOD-STD-2167A <u>Military Standard Defense System Software Development</u> and DOD-STD 2168 <u>Military Standard Defense System Software Quality Program</u> [Ref. 6]) were broken

1

down into six steps. These six steps were software requirements analysis, preliminary design, detailed design, coding and computer software unit (CSU) testing, computer software component (CSC) integration and testing, and computer software configuration item (CSCI) testing.

DOD-STD-2167A (supplanted by DOD-STD-498 in November, 1994 [Ref. 7]), was the primary standard to be used by Department of Defense (DOD) agencies for weapon system software development. The standard was not intended to encourage the use of any particular software development method, instead it was meant to aid the program manager in developing and sustaining quality software. In the latter regard, the standard has been most successful, providing a first step toward a standardized systems engineering approach to software development. [Ref. 6]

A by-product of DOD-STD-2167A, the "waterfall" software development methodology, is a process that applies rigor and a systems engineering discipline to the development of mission critical computer resources (MCCR.) However, it does so at a rather high cost. The process, has increasingly come under fire (from many quadrants) for being inflexible, slow, document intensive, and costly. [Ref. 6]

The near-term fix, DOD-STD-498 Military Standard Defense System Software Development and Documentation, is to be used (at the option of the program manager/contractor), until a suitable commercial/IEEE standard can be developed and implemented. [Ref. 20] With the recent policy shift away from military specifications and standards, MIL-STD-498 will not be required for use by any software development contractor, but will serve as an interim guide for the contractor who

2

chooses to use it. This new standard provides guidance for software development activities and documentation, and allows for greater flexibility in tailoring to meet software development models. [Ref. 7]

The process utilized by the Aviation Mission Planning System (AMPS) Project Office, does not suffer from the many drawbacks associated with the earlier DOD-STD-2167A. This automated mission planning system is being developed for U.S. Army Aviation using a variant of the "spiral" model of software development. This, at once, evolutionary and incremental development process model emphasizes flexibility to changing user requirements, early user participation in the process, decreased documentation requirements/costs, and earlier error detection and elimination (and thus reduced overall costs.)

B.   AREA OF RESEARCH/OBJECTIVES

The area of research for this thesis involves an analysis of the methods employed in the development/acquisition of a mission critical computer resource for an Army Aviation system. Specifically, the research focuses on the methods used in the development of the AMPS, a mission planning system being developed/procured in conjunction with the RAH-66, "Comanche" helicopter.

The process employed represents a significant departure from the "waterfall" process of mission critical computer resources development as reflected in DOD-STD-2167A. This evolutionary model of software/hardware development spans the DOD-STD-2167A and MIL-STD-498 time frames. It will continue to evolve during the

3

movement to commercial practices, and has proven to be extremely flexible, easily manageable, and highly effective.

C. RESEARCH QUESTIONS

1. Primary Question

What are the major features and supporting attributes of the developmental process employed for the Aviation Mission Planning System (AMPS), and how does this process compare/contrast with more traditional developmental methods?

2. Subsidiary Questions

Three subsidiary questions are addressed in this research.

* What are the primary features and attributes (both beneficial and detrimental) of traditional software development methodologies (waterfall, sequential, etc.) that were primarily utilized in conjunction with DOD-STD-2167A?

* Citing recent developments in software engineering, and the directed movement away from the reliance on MIL/DOD-STDs, what are the attributes of more current models employed in the development of mission critical computer resources (MCCR) for major weapon systems?

* What improvements are realized when MCCR is developed under an alternative process such as the evolutionary model employed in the instance of the Aviation Mission Planning System?

D. SCOPE AND METHODOLOGY OF THE THESIS

This thesis investigates past and present development methodologies for mission critical computer resources, highlighting the changes brought about by the paradigm shift from reliance on MIL/DOD-STDs to that of commercial practices. While historical and currently evolving methods were reviewed, the focus of the research are those methods employed in the development of the Aviation Mission Planning System. Utilized in the developing environment of software engineering and the increased use of commercial practices/procedures, these methods are illustrative of the direction in which DOD MCCR developmental efforts are heading.

The primary research question was addressed through a comprehensive investigation of the developmental process for the AMPS. On-site and telephonic interviews were conducted with project management personnel, in St. Louis, Mo., and the AMPS project leader and software engineering personnel within Communications Electronics Command (CECOM), Research Development and Engineering Center (RDEC), Fort Monmouth, N.J.

The first subsidiary question (What are the primary features and attributes, both beneficial and detrimental, of traditional software development methodologies (waterfall, sequential, etc.) that were utilized in conjunction with DOD-STD-2167A?) is answered through a comprehensive review of the standard itself, GAO reports, software management guides, directives and related literature.

The second subsidiary question (Citing recent developments in software engineering, and the directed movement away from the reliance on MIL/DOD-STDs, what are the attributes of more current models employed in the development of mission critical computer resources (MCCR) for major weapon systems?) is answered through both a comprehensive literature search and interviews with several software development/engineering personnel.

The third subsidiary question (What improvements, if any, are realized when MCCR is developed under an alternative process such as the evolutionary model employed in the instance of the Aviation Mission Planning System?) is answered through comprehensive examination of the development process employed by AMPS software development personnel and interviews with project personnel.

A plethora of literature exists that documents the evolution of DOD MCCR development methodologies over the past 25 years. During that time of rigid control and oversight, DOD published numerous documents in the form of handbooks, guides, military specifications, and military standards that delineated MCCR development process details. Additionally, during that same period, the General Accounting Office (GAO) kept itself very busy investigating and documenting examples of process "failure." There is no shortage of sources available to the researcher investigating this area.

More recent developments, for example in hardware and software engineering methods, are also well documented. Data from this area were collected from the

literature and also through on-site or telephonic interview with project/program personnel and industry experts.

Lastly, there is little "published" information available which references the developmental process employed for the AMPS. Therefore, the bulk of the data in this area of research were collected exclusively through an on-site visit and on-site and telephonic interviews.

Several research limitations exist that narrow the scope of this effort. Uncertainty of the impact of new laws, regulations, and directives is first and foremost. An example of this is DoD's transition from DOD/MIL-STD (e.g., MIL-STD-498) to commercial standards. This transition is only beginning to occur within MCCR development agencies as this research is conducted. There is no way to predict how "the dust will settle" when this transition is complete.

Additionally, the AMPS is not yet a "finished product" from an acquisition standpoint. The system is just now transitioning from prototype, and is far from being complete and ready for Army-wide fielding. Again, there is no way of accurately predicting the future attributes of the evolving developmental process employed for this system's software. A myriad of factors will influence the degree of success realized as the process transitions from a prototyping to a production effort.

Time available to conduct the research, and geographic distance of researcher from developer also presented challenges. These were largely mitigated through an aggressive and highly organized approach to the literature search, on-site visits, and interviews. In this regard, a questionnaire was employed to gather data about the

7

AMPS software development process. This questionnaire was sent as a read-ahead packet to selected personnel within the AMPS project and development offices. The intent of the questionnaire was not to gather statistical data, but to highlight areas of focus for the researcher's subsequent data-gathering visit. Written and/or verbal responses were provided by both offices (i.e., product and development) for all questions. The questionnaire is included as an Appendix to this thesis.

E.   BENEFITS OF THE STUDY

Software is on the critical path for all major weapon systems under development today. Successful software development processes must be sought out, emphasized, studied, and further developed. The advantages and disadvantages of these methods must be determined and, if appropriate, these methods must be applied to future undertakings. This thesis studies in detail a true rarity in the realm of military software development: a developmental process that ensures flexibility to changing requirements, responsiveness to the end-user, less documentation, and higher quality (less errors) at reduced cost.

F.   ORGANIZATION

This thesis consists of five chapters. Chapter II establishes the background and framework for the investigation of the area of interest. This chapter provides a brief insight into MCCR development problems throughout its history. DOD-STD-2167A and DOD-STD-498, and their accompanying developmental methodologies, are

discussed to provide the reader with a "backdrop" for the present evolution in MCCR developmental methodologies. Lastly, this chapter briefly introduces the reader to the AMPS.

Chapter III begins the in-depth investigation of the AMPS software development process. The intent of this chapter is to present the prevailing attributes of the developmental process employed by the AMPS project personnel, as the process evolves from prototype to production software development activities.

Chapter IV presents an in-depth discourse on several areas worthy of additional emphasis. Additionally, potential problem areas are highlighted and possible risk-mitigation strategies are explored.

Chapter V summarizes the issues discussed in the previous chapters. Results of the discussions/interviews are used to draw conclusions from the issues presented and make recommendations. Finally, this chapter explores directions for continued research/study.

II. BACKGROUND

This chapter provides the background and framework for the investigation of the area of interest. It begins by providing a brief insight into MCCR developmental challenges and problems of the last 30 years. Then, bringing the reader more up to date, two key DoD standards (DOD-STD-2167A and DOD-STD-498) are reviewed. These standards have more recently been instrumental in shaping MCCR development for major weapon systems. The predominant methodologies that emerged from the application of these standards are also discussed. Process attributes, both positive and negative, are the focus of this discussion. The recent shift away from military specifications and standards is then briefly touched upon, ending in a discussion of the current MCCR developmental environment. Lastly, the chapter introduces the reader to the Aviation Mission Planning System and the methodology employed in the development of this system.

A. GENERAL

Over the last 35 years, the development and fielding of U.S. major weapon systems has undergone a revolutionary transformation. Since the 1960s, the weapons being developed, produced, and maintained in this country have come to rely heavily on computer resources (hardware and software). Technological advancements in these areas and the commensurate growth in weapon systems capabilities have been mind boggling. Unfortunately, the Government's ability to effectively and efficiently

procure these systems has not advanced at the same pace. Alarmed by the marked increase in procurement difficulties and failures, the Government has invested much time and energy uncovering the root causes. In this vein, the General Accounting Office (GAO) has been very successful in determining some of the reasons for these shortcomings. Time and again, mission critical (specialized) computer resources are determined to be at the heart of the problem. The Department of Defense's ability to effectively and efficiently develop and contract for these items will directly affect the future readiness of the Armed Services. In today's world of rapid technological advances and simultaneously shrinking budgets, the need for this ability is further amplified. [Ref. 6]

Digital computers and their accompanying software were in their infancy in the 1950s and only just began appearing in weapon systems in the 1960s. The F4 "Phantom" was the last jet fighter aircraft to rely purely on "hardware" control linkages (push-pull rods & hydraulic actuators) [Ref. 1]. As shown in Figure 1, this contrasts sharply with the 5-7 million lines of code (software) that will be required to keep the Air Force's Advanced Tactical Fighter (ATF) aloft. [Ref. 1]

Today, all major weapon systems are dependent on computers and their software. This does not only pertain to the advanced flight control programs designed for use in military aircraft. In fact, every major weapon system in the inventory relies on computer software in one way or another to accomplish its mission.

```
WEAPON          LINES OF SOFTWARE CODE*
 F-4               0(VIRTUALLY)
 F-16D           236,000
 C-17            750,000
 B1-B            1.2 million
 ATF             5-7 million
 SDI             25 million (est.)

* Lines of code are often used as a major factor for
describing the complexity of a software program.   In
addition, it should also be noted that a doubling of the
lines of code does not normally equate to a doubling of
complexity.  Rather, a more likely result is a program as
many as 10 times more complex.
```

Figure 1. Weapon System Software Complexity Comparison [Ref. 1].

B.   MISSION CRITICAL COMPUTER RESOURCES (MCCR) PROLIFERATION

In many of today's highly advanced systems, there are numerous computer

systems organic to the weapon. Additionally, there is a wealth of other software

required to support the majority of today's fielded systems.   Examples of the plethora

of software include: 1) software used in support of crew training, 2) battle

management software, 3) maintenance trainer software, 4) crew training software,

5) mission preparation software, 6) data reduction software, 7) scenario analysis

software, and 8) automatic test equipment/test program set software.

Digital (computer) systems are now the heart and soul of all new weapon

systems. The flexibility afforded by digital systems cannot be remotely approached

by analog systems. In essence "hardware" is replaced by "software" whenever feasible. This trend will continue into the foreseeable future. [Ref. 6]

C. CURRENT MCCR STATE OF AFFAIRS

Examination of the current state of affairs with regard to MCCR highlights some revealing, and sometimes, undesirable attributes: [Ref. 6]

(a) Most new weapon systems are extremely complex. This is due to a combination of several factors [Ref. 6] :

- extremely demanding requirements (Which incidentally, tend to both "grow" in scope and "shift" in focus);

- tight schedules and even tighter budgets, which tend to negate elegant and simpler solutions;

- and too many contractors not fully skilled in software engineering techniques.

(b) Most systems are delivered late, have cost overruns and rarely meet performance requirements upon initial delivery. These systems are often ridiculously expensive to maintain.

Though not fully responsible for these cited shortcomings, mission critical software is generally recognized as a major contributor to these problem areas. In short, in modern major weapon procurement, software development/procurement is always on the critical path. In a article written by James Kitfield, the author cites a speech by Air Force General Bernard Randolph, chief of Air Force Systems Command. Characterizing software as the "Achilles heel" of weapons development he

said, "On software schedules (development) we've got a perfect record: We haven't met one yet" [Ref. 16].

The costs associated with the above difficulties are staggering. In a General Accounting Office (GAO) report on the subject of software costs, it was revealed that DoD does not truly know what it spends on this critical technology. The estimates of software costs range from $24 billion to $32 billion annually, about 8 to 11 percent of Defense's total budget. The report went on to stipulate that these estimates are rough at best, because DoD has not identified or tracked software costs as a discrete item in its weapon systems development programs. [Ref. 10]

In its defense, DoD long ago recognized serious shortcomings associated with its procurement process for MCCR. In this regard, it has published numerous handbooks, specifications, directives, and standards aimed specifically at process improvement. Two of the more far-reaching standards, and their associated methodologies, are presented here.

D. DOD-STD-2167A: PART OF THE PROBLEM ?

DOD-STD-2167A, implemented 29 February 1988 [Ref. 20] is a process standard for the development of mission critical computer software. It evolved from an earlier software development standard, DOD-STD-2176. Intended to mitigate much of the inflexibility, limitations, and onerous requirements of the earlier standard, DOD-STD-2167A, unfortunately falls short of this goal.

15

The standard is designed to apply a systems engineering framework to the development process. As such, it defines development and management activities, the phases of development, documentation, and prescribed audits. It was designed to aid the program manager in the development and sustainment of quality software. While this standard does not mandate a particular developmental method, it was often believed to endorse a "waterfall" methodology because this method was used as an example throughout the document [Ref. 7]. The standard is designed to be "tailored" to the specific program or project through the deletion, modification, merging, addition or qualification of requirements. Specific guidance on DOD-STD-2167A tailoring is detailed and published in MIL-HDBK-287.

The six major phases, or activities detailed by the standard are:

- Software Requirements Analysis

- Preliminary Design

- Detailed Design

- Coding and CSU Testing

- CSC Integration and Testing

- CSCI Testing

These steps or phases are to be repeated in sequence as many times as necessary in the development cycle of the software. The sequential nature of the framework mandated by the standard (analyze-design-code-test), resulted in the "waterfall" software development methodology or paradigm. [Ref. 34]

16

1. The "Waterfall" Model of Software Development

The "waterfall" software development model ensures that the "steps" of the development are performed in the specified order, as depicted in Figure 2 [Ref. 34]. All requirements are defined up front and comprehensive reviews are used as "gates" that must be passed through to proceed to the next step in the process. The model mandates that all program design be complete before any coding begins.

Some problems with this method of development can be readily seen. To begin with, the initial requirements analysis is rarely adequate. Users and developers usually come to the drawing board with an unrealistic, static view of required
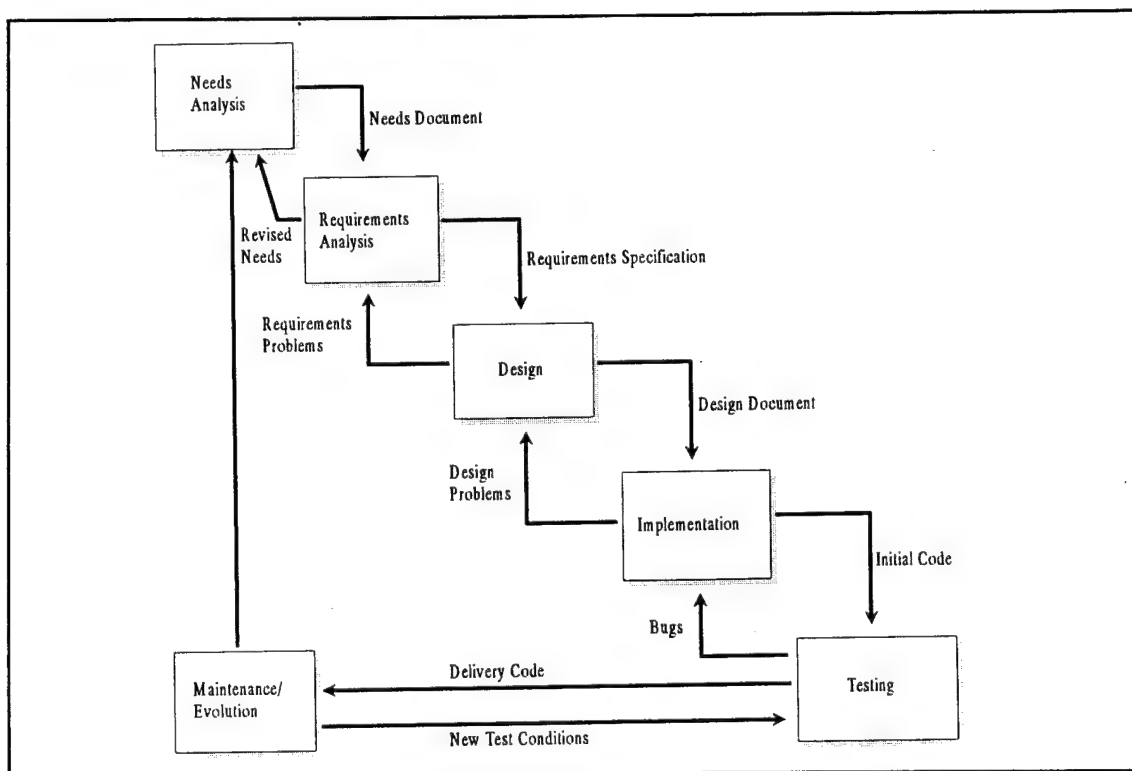
Figure 2. Waterfall Model of Software Development [Ref. 34].

attributes. This approach is rarely successful, as user desires change during development or are changed by outside forces/factors. Just as often, customers do not have a good feel for what it is they really need (or if they do, they cannot properly articulate that need). Thus the requirements specified are often incomplete, inconsistent, or not implementable. Today's movement toward evolutionary requirements (i.e., requirements that evolve through the acquisition process) is not readily supported by the "waterfall" methodology [Ref. 11].

The "waterfall" model also fails to take parallel and concurrent development activities into account. Realistically, development activities do not occur in sequence, and the DOD-STD-2167A reviews and audits (the gates) follow the same "lock-step" sequential logic of the activities flow. Essentially, the "waterfall" label is a misnomer, as real world development activities follow no prescribed sequence, instead moving up and down the "waterfall" as need, and a changing situation dictate. [Ref. 34]

Documentation requirements are also excessive for DOD-STD-2167A [Ref. 20]. Adequate documentation, that which sufficiently ties software requirement objectives with standards for performance, software design, test plans, software code, and the results of software test and evaluation, is clearly needed [Ref. 20]. DOD-STD-2167A however, mandates no fewer than 19 variants of specification, document, plan, description, list, code, or report [Ref. 20]. Though the standard mandates no particular development process, the documentation requirements are not subject to "tailoring." They are directly tied to the major development activity being undertaken

18

and are checked for adequacy (and approved/disapproved) during the accompanying review audit. [Ref. 20]

2. The "Spiral" Model of Software Development

The "spiral" model or methodology is another process variant for software development that was supported, though to a much lesser extent, by the framework established by DOD-STD-2167A. This method better recognizes the non-sequential nature of software development activities, promoting an iterative build of the system. Additionally, the model allows the developer to better track the growth of information (e.g. the status of phase activities) about the system , allowing for the fact that knowledge grows at a slower rate than the system [Ref. 34]. In this fashion, the model provides a risk reduction approach to software development [Ref. 8].

The spiral model combines basic waterfall and evolutionary/incremental prototype approaches to software development. The basic waterfall "building blocks'" (e.g., Preliminary Design Review, Detailed Design, Critical Design Review, etc.) are followed sequentially to deliver an initial operational capability (IOC). A risk analysis phase evaluates support and maintenance issues, the product is updated, demonstrated, and validated. The product then progresses through an "updated" waterfall development process, and a final operational capability (FOC) product is delivered. This process may occur several times, hence the "spiral" label. A depiction of the "spiral" (e.g. Barry Boehm's spiral model) model of software development appears in Figure 3. [Refs. 34, 8]

It has been noted that there are problems associated with the "spiral" model as well. Three significant issues that have been raised are: 1) Who defines the end of the "spiral" process, and what should the end of the process coincide with? For example, is the end of the process defined by the availability of funding, the continued need for system requirements (and system upgrades), or when some established "goal" has been reached? 2) How do Government managers evaluate the process and its products in a systematic manner? 3) The "deliverable" consists of software code and documentation, yet this process (as it has been predominantly applied) significantly slights documentation. [Ref. 35]
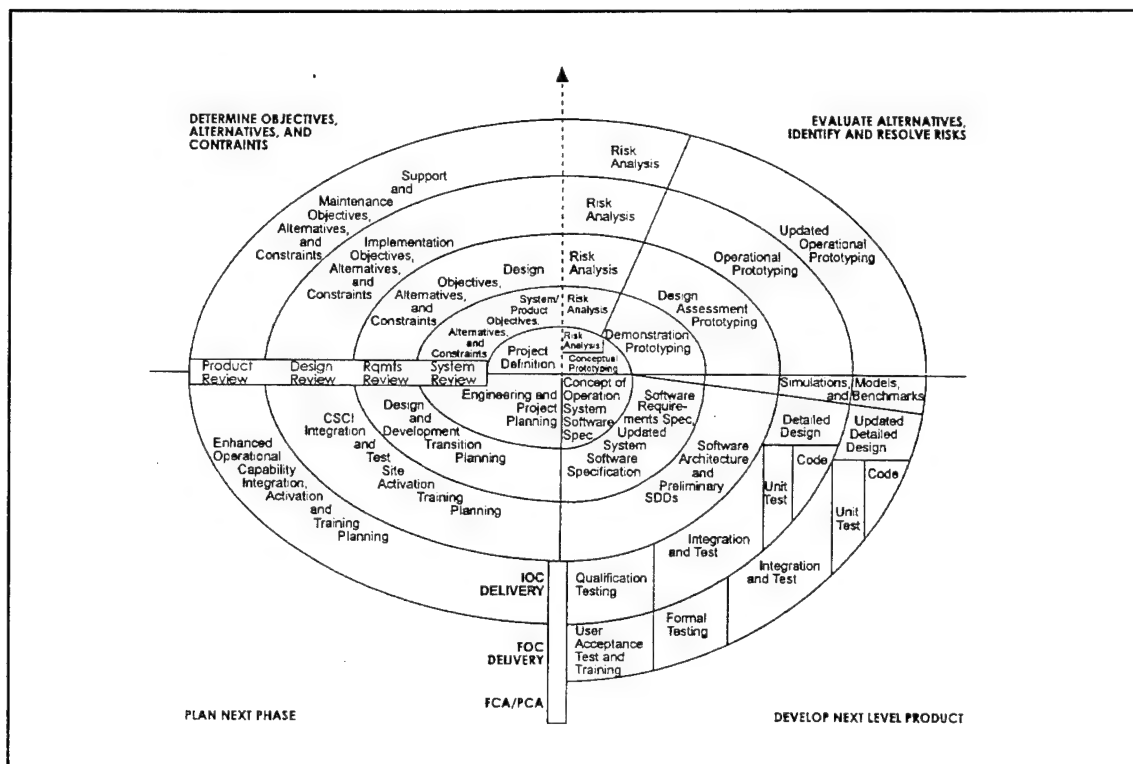
Figure 3. The "Spiral" Model [Ref. 8]

E.   MIL-STD-498: THE INTERIM FIX

On 8 November 1994, MIL-STD-498 replaced DOD-STD-2167A and other related standards [Ref. 20]. This action was largely taken as a stop-gap measure. It was designed to mitigate some of the deficiencies of previous standards, and at the same time serve in the interim between the SECDEF directed abandonment of DoD and MIL standards and the adoption of commercial standards, practices, and procedures [Refs. 20 22]. There is no current civilian standard that encompasses all the aspects of weapon system software development

The intent of MIL-STD-498 is not so much to depart from the tenants of DOD-STD-2167A . Instead it is intended to promote tailoring (including documentation) and increase the inherent compatibility of DoD software development with various development models. The standard itself includes specific guidance on tailoring and includes examples that accommodate "grand design" (or "waterfall"), **"evolutionary"** and **"incremental"** models, as well as the use of **prototyping** [Ref. 8].

Essentially, the standard is a "loosening" of DoD developmental requirements. It provides a logical step toward promoting an environment that more closely represents that which will be experienced when DoD fully transitions to commercial standards, practices, and procedures.

In light of the SECDEF's new policy, it is not surprising that MIL-STD-498 does not require the use of any DoD or military standards. Further, it provides a reference that relates major development activities to recognized commercial industry

21

standards [Ref. 7, 20]. The data item descriptions (DIDs) for the standard encourage

the use of compatible commercial items that meet contract requirements. [Ref. 7, 20]

Further moving toward a commercial software development environment,

MIL-STD-498 loosens the audit/review framework, allowing these events to be tailored

to the development process being employed. Lastly, though MIL-STD-498 DIDs

require some 22 documents (14 being common to those required by DOD-STD-

2167A), no specific format is mandated for those documents, and only those required

to support a particular development are specified. [Ref. 7, 20]

The following sections discuss three methodologies covered in MIL-STD-498:

the evolutionary model, the incremental model, and prototyping. [Ref. 8]

1.   The Evolutionary Model

Evolutionary development in many ways is similar to the spiral methodology.

However in an evolutionary process there is far less emphasis on the execution of the

sequential building block activities called for by the waterfall and spiral models. This

model requires the development of a fully documented and operational initial product.

Here, the emphasis is on the development of a flexible, modular, operational "core"

product, and the subsequent refinement of the product through later versions/builds.

The core includes provisions for future functionality and requirement changes. Figure

4 depicts Pressman's interpretation of the evolutionary model. [Ref. 8]

Distinct from the previously discussed models, the evolutionary model

emphasizes early and constant user feedback by calling for the development of

demonstrable software increments. Figure 5 depicts user involvement throughout the

evolutionary development process [Ref. 8]. In addition, evolutionary developments

are conducted as a planned progression towards an ultimate initial functional

capability. Hence, there is no danger, as in the spiral model, of not knowing when to

cease the development process. [Ref. 35]
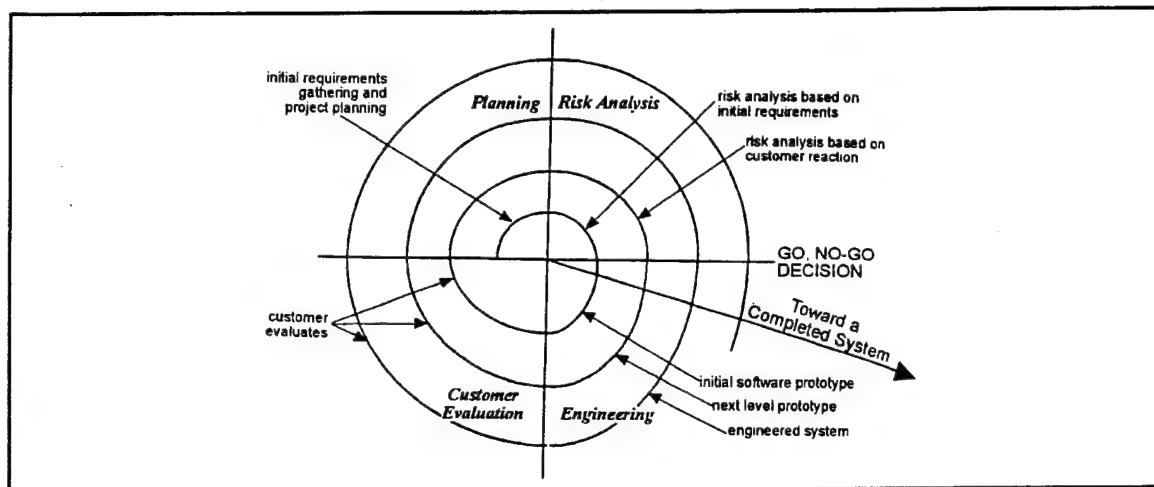
Figure 4. Pressman's Evolutionary Model [Ref. 8].
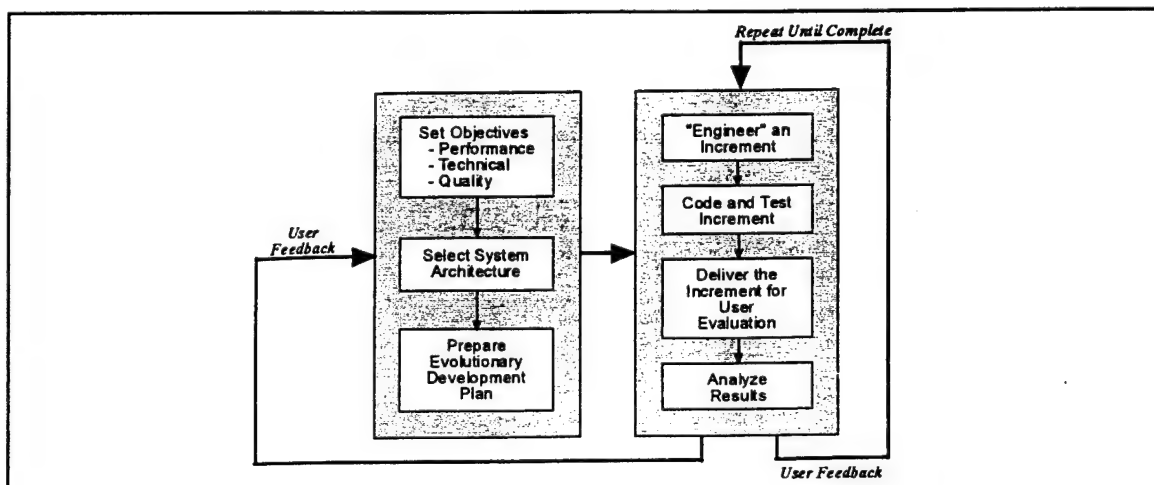
Figure 5. Evolutionary Development - User Involvement [Ref. 8].

23

2. The Incremental Model

The incremental model calls for the development of software in groups of functional capabilities, or subsets. Here, the total software package is divided into increments that are developed in phases over the total development cycle. This allows employment of part of the product before final completion. [Ref. 8]

This development strategy is characterized by the *build-a-little, test-a-little* approach, where an initial functional subset is delivered and subsequently augmented or upgraded until the ultimate functional capability is achieved. Figure 6 depicts the incremental development process. [Ref. 8]
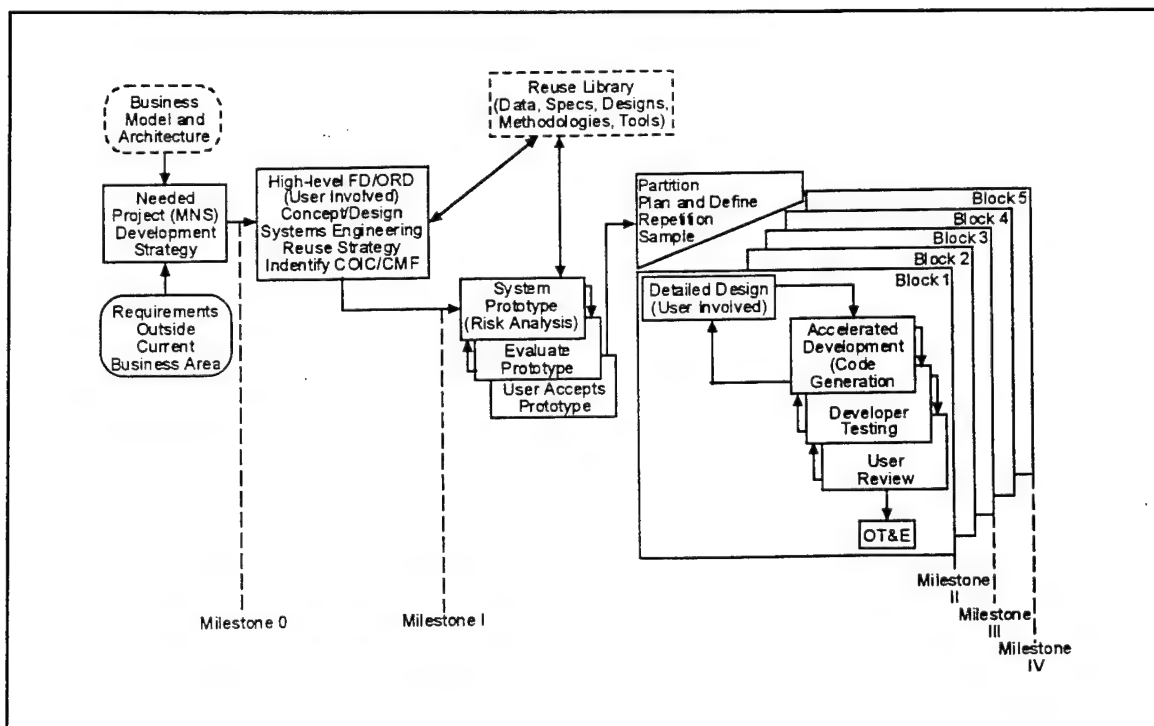
Figure 6. The Incremental Development Process [Ref. 8].

3. Prototypes

This type of development effort is characterized by the timely development and deployment of functional products that allow the user the opportunity to compare alternatives and better articulate requirements. Though the technique may be used throughout the life cycle process, prototypes prove especially advantageous when employed during Concept Exploration/Definition and Demonstration/Validation phases. Here, they assist greatly in requirements definition. [Ref. 8]

Typically, the focus of the prototype effort is on a functional product (e.g., functional code). Design architecture, documentation, configuration management, and other procedures take secondary precedence to those efforts/procedures required to rapidly produce a useable product. [Ref. 8]

F. PERFORMANCE SPECIFICATIONS & COMMERCIAL STANDARDS

The Secretary of Defense, William J. Perry's 29 June 1994 memorandum entitled, "Specifications and Standards -- A New Way of Doing Business," [Ref. 22] clearly charts the course for all current and future development/acquisition programs. Military specifications and standards are, except for instances of waiver, a thing of the past. Performance specifications will be used when purchasing new weapon systems or major modifications, upgrades to new systems, and non-developmental and commercial items. This applies to systems in all Acquisition Categories (ACAT) [Refs. 20, 22]. Non-Government standards are to be used if performance specifications

are not practical. In the event that non-Government standards are unacceptable or the performance specification or non-Government standard is not cost effective, an applicable military specification may be used once a waiver is approved. [Ref. 22]

The intent of the guidance is three-fold. First, the new methodology is meant to make commercial state-of-the-art technology more accessible to DoD. In addition, it facilitates a teaming of industry and defense development and manufacturing processes and facilities. This will enhance the development of dual-use technologies while expanding or strengthening the defense-industrial base. Lastly is the commensurate reduction in acquisition costs that DoD stands to realize if this teaming paradigm can be implemented. [Ref. 20, 22]

Thus far, the intent of this chapter has been to provide insight into the environment in which current systems (including the Aviation Mission Planning System) are being developed. The last section of this chapter will familiarize the reader with the attributes of the Aviation Mission Planning System (AMPS.)

G.   THE AVIATION MISSION PLANNING SYSTEM (AMPS)

This section provides an overview of the Aviation Mission Planning System (AMPS.) It includes the objective, capabilities, and configuration of the system. The intent is to provide a backdrop for the system software development process investigation covered in the following chapter.

1. The AMPS Overview

Tactical Army aviation mission planning is a complex process that
encompasses multiple, diverse tasks. The planning process encompasses such tasks as
the tactical route, communications, and crew and aircraft configuration planning.
These tasks are based on inputs such as the enemy and friendly situation, flight
hazards, weather, radio and personnel data, and aircraft specifics. [Ref. 26]

Tactical mission planning has traditionally been a manual exercise performed
by planning teams. The process, depending on the complexity of the mission, can be
very time consuming, inefficient, and error prone [Ref. 26]. Typically, a planning
team or "cell" is composed of key members of the tactical unit (e.g. section, or platoon
leaders, the commander, and/or other key personnel.) Their time and energies leading
up to actual mission execution are extremely valuable. Any system that will increase
their efficiency is needed and highly desirable.

The AMPS provides this capability, automating mission planning tasks and
freeing up key personnel to coordinate, communicate, and finally check mission
specifics and instructions. The overriding objective of the AMPS is clearly delineated
in the following paragraph drawn from the AMPS User's Manual [Ref 26]:

> The objective of the AMPS is to provide aviation mission planners with
> a tool to automate their mission planning tasks, which may otherwise be
> more labor intensive, time consuming, and error prone. The AMPS
> utilizes a menu-driven user interface that includes a combination of both
> tabular and mapping overlay data entry thereby allowing the mission
> planner to effectively develop a mission in a productive manner.

27

2. The AMPS Mission Process

Initially, the AMPS is used by the mission planner to develop a mission through a logical progressive sequence of operations [Ref. 26]. These operations aid the planner in preparing essential data. This includes tactical route planning, crew and communication planning, and aircraft configuration. These, as mentioned above, are based on inputs such as friendly and enemy situation, weather, flight hazards, communications and personnel data, and aircraft specifics (e.g. type, category, etc.). [Ref. 26]

In addition to providing mission planners with an automated tool for mission planning, the AMPS is designed to provide for the transfer of mission data to the aircraft via a Data Transfer System (DTS) [Ref. 26]. This sub-system includes a Data Transfer Cartridge (DTC) that contains the mission data files created by the planner(s). The DTC is easily moved from the AMPS station to the (pre-mission) aircraft where the data are transferred. During the mission the DTC records mission related data (e.g. airspeed, altitude, aircraft warning messages/advisories, and engine history.) Post-mission, this device is transported back to the AMPS device and the data are down-loaded for manual analysis and mission back-brief [Ref. 26]. The mission process is graphically depicted in Figure 7. [Ref. 26]
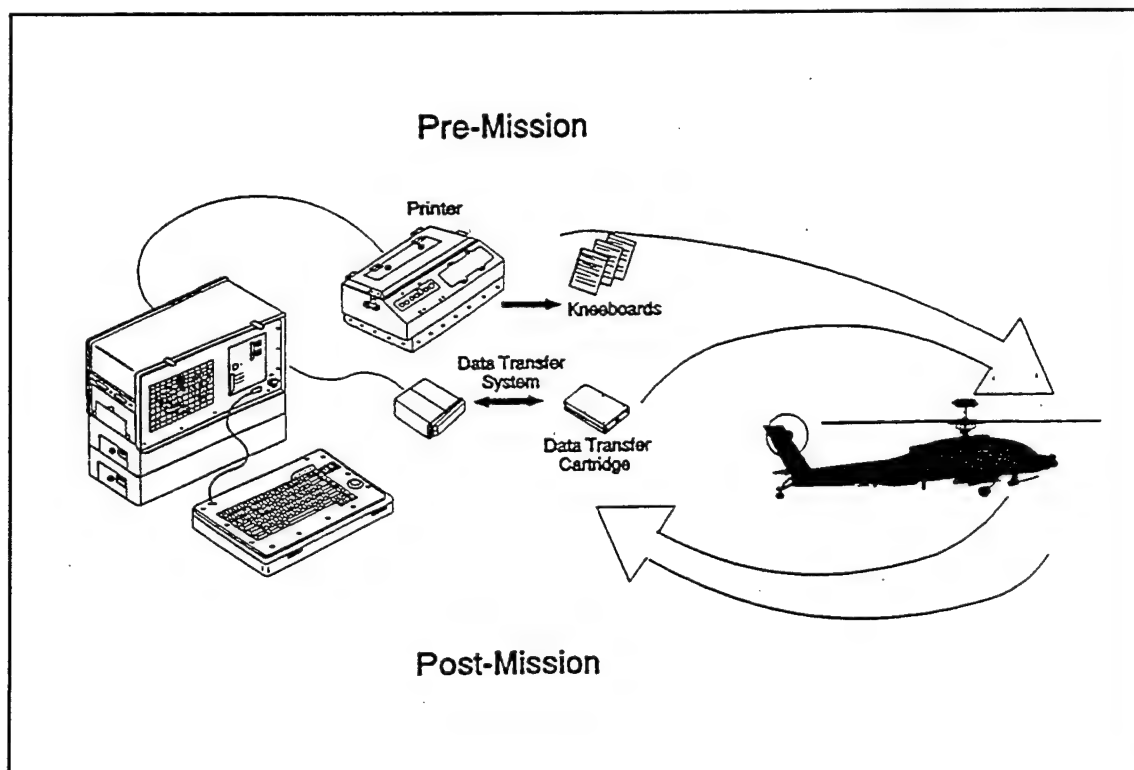
Figure 7. The AMPS Mission Process [Ref. 26].

3. The AMPS Configuration (Hardware)

The AMPS hardware configuration consists of a computer unit and associated

peripherals. It is depicted in Figure 8 [Ref. 26]. The computer unit, designated the

Lightweight Computer Unit (LCU), version 2, is a ruggedized portable computer that

consists of the following [Ref. 26]:

- 33 Megahertz (MHz) 80486 32-bit processor with 32 Megabytes (MB) of main memory and an embedded floating point processor.

- Detachable, 82-key enhanced keyboard with 101-key functionality and an embedded trackball with three control switches.

- 640 x 480 pixel 9.4" diagonal color liquid crystal display (LCD) screen.

29
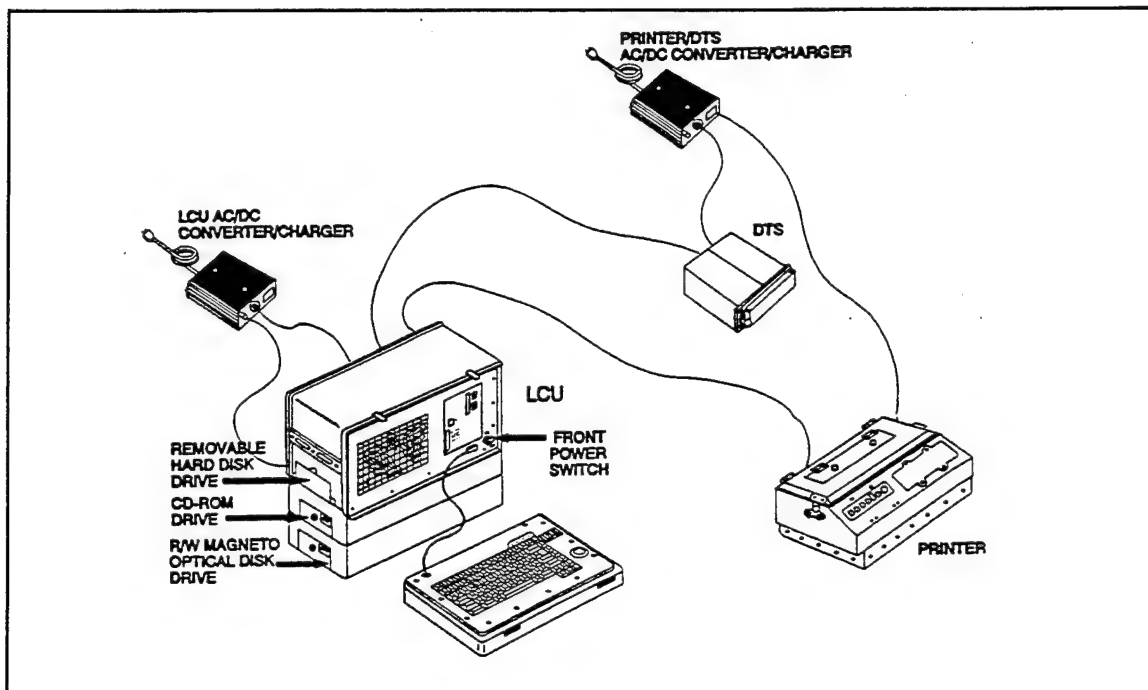
Figure 8. The AMPS Hardware Configuration [Ref. 26].

- 500 MB removable hard disk drive.

- 3.5" 1.44 MB floppy disk drive.

- MIL-STD-1553B bus compatible input/output (I/O) interface.

- 9600 bits per second (bps) modem .

External peripheral devices include the following :

- Compact Disc-Read Only Memory (CD-ROM) drive.

- Read/Write Magneto Optical (MO) disk drive.

- Lightweight, portable dot matrix printer.

- Data Transfer System (DTS), MU-1004/1005, including Data Transfer Cartridge (DTC).

- LCU power source: 110/220 volts, alternating current (VAC), 50/60 Hertz (Hz) with an alternating current (AC)/direct current (DC) converter/charger or 24 volts, direct current (VDC) rechargeable battery pack for 2 hours operation, or 28 VDC vehicle battery with AC/DC converter/charger.

- Printer/DTS power source: 110/220 VAC, 50/60 Hz, AC/DC converter/charger or 28 VDC vehicle battery with AC/DC converter/charger.

4.   The AMPS Software Environment

As stated in the User's Manual, the software environment is completely

transparent to the user. The system is delivered as a "turn-key" device. The AMPS

software is contained in the AMPS system, and does not require user intervention for

loading. [Ref. 26]

The development process for the system software is the main focus of this

research. Of general interest is that this process began during the DOD-STD-2167A

time frame. Its evolutionary development has continued under the requirements of

MIL-STD-498 and will continue to further evolve during the transition to performance

specifications and commercial standards.

H.   SUMMARY

This chapter provided a general background and framework for the study of

MCCR development/acquisition. Problems associated with the procurement of MCCR,

specifically software, were highlighted. The discussion of DOD-STD-2167A, MIL-

STD-498, and SECDEF's directive to use performance specifications and commercial

standards was intended to further define/describe the environment in which the

Department of Defense develops and acquires mission critical software. A brief

discussion of software development models familiarized the reader with the methods associated with the application of the various standards. Lastly, an overview of the AMPS was provided, to include basic system attributes, mission process, and system configuration.

The next chapter is a case study of the unique development process employed for the AMPS software. The chapter begins by examining the development of the prototype-system software for "Interim AMPS," then turns to the study of the evolving process for the development of the "AMPS" production software.

III. THE AMPS SOFTWARE DEVELOPMENT PROCESS: A CASE STUDY

The intent of this case study is to document what to date has been a successful DoD sponsored software development process. The focus of this effort, the AMPS, is being developed for the Program Manager's Office, Aviation Electronic Combat (PM-AEC), Program Executive Office (PEO), Aviation, St. Louis, Mo. This effort is being conducted in-house by the Command and Control Systems Integration Directorate (C2SID) at the Communications-Electronics Command's Research, Development and Engineering Center (CECOM RDEC) at Fort Monmouth, New Jersey. [Ref. 25]

The goal is to examine several key areas of the software development process, and document the "successful" process attributes and value-added efforts (and effects) implemented by the software development team. The principal areas of interest include process generalities (e.g. methodologies and techniques employed), and several specific aspects. These include coding and documentation practices, configuration management policies and procedures, test and evaluation (T&E), and independent validation and verification (IV&V.) Examination of the applied management principles (e.g. metrics, tools, the software engineering environment, etc.) is the final area of interest.

Though quite successful through the software prototyping stages, the development process (from a production software perspective), is not currently considered mature and is somewhat unstable. An accurate description of the development process for the AMPS' software is one in **transition.** [Ref. 25]

33

For reasons that will be discussed later, the Government in-house developer, C2SID, has been tasked to transition its organization, mind-set, and software developmental process from one adept at prototyping efforts, to one capable of developing quality, deployable software. The management (PM-AEC) and development team's (C2SID) efforts to implement this process transition is the main focus of this case study. [Ref. 25]

Though there is no actual "clean break" associated with the shift in emphasis from prototype software development efforts to the development of production software, it is easier to visualize the two process variants as separate and distinct. In reality, prototyping efforts, and methods continue for the AMPS software development program. However, it also holds true that process change and improvement initiatives are being considered and implemented by management and developer alike. This case study treats the over-arching process as two separate and distinct processes: the software prototype process employed for the "Interim AMPS"; and the production software process that is being established for "The AMPS." [Ref. 25]

The chapter begins with an examination of the AMPS development process as it existed during the previous "Interim AMPS" prototyping effort. A general discussion of software-prototype process attributes is also made. The Interim AMPS prototype-process discussion focuses on the following areas: 1) coding and documentation practices, 2) configuration management policies and procedures, 3) test and evaluation and IV&V, and 4) the application of management principles.

The paradigm shift (i.e., from prototype to production software development) and its cause, are then outlined. Next, the chapter turns to an examination of the current process (albeit in a transitory state). The areas of process improvement implemented (or being developed/considered) by the AMPS management/development team are highlighted. This examination will focus on the same areas examined for the Interim AMPS prototyping oriented process.

This process examination serves as lead-in to Chapter IV in which the researcher calls attention to several areas worthy of additional emphasis or consideration, and highlights some specific areas of concern for the program.

A. "INTERIM AMPS" : THE PROTOTYPE SYSTEM

Since its inception, the development of the AMPS system has been closely tied with the development schedules of several aircraft (e.g. OH-58D, AH-64A Mod/D, UH-60 A/L,CH-47D, and the RAH-66, Comanche). Citing an immediate need for mission planning capability to support near-term aircraft development requirements, C2SID, of CECOM RDEC was tasked to develop an "interim" system that would provide mission planning functionality, pre-flight avionics systems initialization, and data load capability for the OH-58D (Kiowa Warrior), AH-64D (Longbow), CH-47D (Chinook), and AH-64A Mod (Apache) systems [Ref. 24]. The initial or "Previous Strategy" for AMPS development/acquisition is depicted in Figure 9 [Ref. 25].

Interim AMPS has been characterized as a "proof of principle" system [Ref. 3]. However "prototype," "rapid prototype," or "demonstration" label is more appropriate. The original acquisition strategy called for the production system to be developed by a commercial contractor. The intent was to develop the AMPS production system around the already fielded Air Force Mission Support System (AFMSS), a system possessing much of the same core functionality thought needed for the AMPS [Ref. 25]. Interim AMPS was seen as a risk reduction and technology insertion vehicle. It was hoped that through the continued development of Interim AMPS, not only would near-term capability requirements be met (e.g. mission planning, data load, avionics initialization), but user requirements would be more
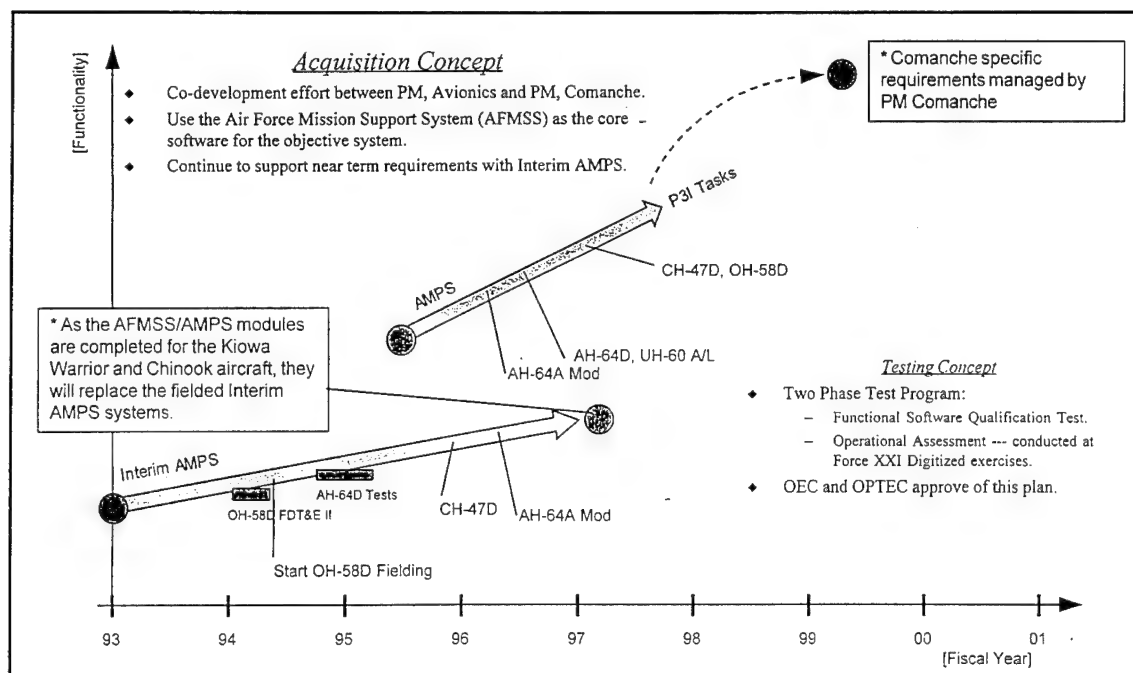
Figure 9. The Initial or Previous Strategy for AMPS Acquisition [Ref. 25].

adequately defined when the transition was made to the production "AMPS" development. [Ref. 24]

1. The Software Prototype Development Process

Interim AMPS software development at CECOM, C2SID possesses the attributes of a prototype/rapid prototype process. As with any prototype effort, the main thrust of the work is to further define and understand system requirements, test alternative approaches to system design, and generate and elicit user feedback and buy-in [Ref. 8]. Inherent in most prototyping efforts is the bypassing of normal configuration management, interface controls, technical documentation and supportability requirements [Ref. 15]. Indeed, to incorporate quality control and assurance (testing) and supportability issues (e.g., technical documentation) beyond the cursory level, would negate the benefits of prototyping [Ref. 8].

The following information was gathered during the August, 1995 AMPS Working Group Review and through on-site interviews with C2SID and Software Engineering Directorate, Avionics (SED-AV), CECOM personnel. The researcher found that the policies, procedures, and practices employed for the development of the Interim AMPS, in the areas of coding, documentation, configuration management, test and evaluation (and IV&V), and applied management principles, were what would be expected for a prototype effort [Ref 15].

a. Coding/Technical Documentation

Coding, in the "ANSI C" language is carried out by several programmers (approx. 8). Responsibility for modules is "broken out" to specific programmers, the synthesis of which is predominantly the responsibility of the chief programmer/software engineer (S.E.) For this project the S.E. has also taken on coding responsibilities for some of the modules. Additionally, the S.E. was the author of much of the system's early code. Thus, the bulk of the expertise for the overall software code-package is concentrated in one position, that of the S.E. [Ref. 31]

This is somewhat "dangerous" from a supportability/maintainability standpoint. It has been noted by development personnel, who have further encouraged a "breaking out" of code structure and programmer intent to supporting programmers. Though in the past this process attribute has been seen as "a concern" to the development team, it is not unusual for a prototyping effort. For the Interim AMPS the system requirements development, limited functional capability, and interface development have taken precedence over supportability and maintainability issues. In the view of the system developers, the primary objectives of Interim AMPS have clearly been met. [Ref. 31]

Technical documentation records the engineering process and helps software maintainers and other engineers understand the code developed by others. It is often slighted in a software prototyping effort. For prototyping efforts, coding standards (e.g. DOD-STD-2167A, MIL-STD-498, etc.) are usually avoided. Requiring such standards would adversely impact the benefits of prototyping [Ref. 8]. For the

38

Interim AMPS much of the code, especially that code written in the early stages of the software project, is not documented adequately for supportability and maintainability. Again, this is not unexpected and would not normally be considered problematic for a prototype development. However, the fact that there is now a transition of the prototype AMPS to a production variant has recently raised concerns. [Ref. 31]

b. Configuration Management

There are three primary reasons for software configuration management: 1) identification, 2) communication, and 3) cost control. These apply to any software development effort. However, normal configuration management procedures typically suffer in a software prototyping environment [Ref. 15]. The true objectives of a prototyping effort take priority. The desire to balance the need for some control with the unproductive effects of over-control must be considered for a prototyping project. [Ref. 19]

For the Interim AMPS, the configuration management (CM) methods employed to date appear to have been largely effective in the area of software identification. However, in the areas of communication and cost control these methods are not as effective [Ref. 31]. As for software identification, suffice it to say that it is evident that through a combination of naming (e.g. "Interim AMPS" or simply "AMPS") and versioning (e.g. 3.0, 3.1, 3.2, etc.), obsolete, current, and future software have been clearly identified and controlled. This has been an especially

39

important factor for participating Interim AMPS program personnel who are geographically dispersed.

As for the other two areas facilitated by CM, (communication, and cost control), results have been less effective [Ref 17]. Communication between programmers about changes to the software does not appear to be significantly effected by CM measures employed for the Interim AMPS. This can be partially attributed to the development environment. The limited number of AMPS programmers, and the fact that they work together in the same general office space, on the same shift (e.g., day), serves to significantly mitigate "communication" problems for software programmers. However, when programmers make changes to the software (e.g. enhancements or problem corrections) "on the fly," or from remote locations (e.g. home/travel via modem) an opportunity for communications breakdown exists. A concerted effort has been made to reduce these instances, regardless of the "need for speed" and the prototyping nature of the Interim AMPS. [Ref. 32]

Programmers are but one entity that require timely information on a current software version. The remainder of the development team (to include, user, test and evaluation and IV&V personnel at SED-AV, CECOM, and program management personnel at PM-AEC (St. Louis)), also need up to date information on the capabilities and limitations of the most recent software version. [Ref. 31]

A key factor for cost control, as applied to software CM, is that some identified problems in the development may not need to be immediately fixed. In fact some may be prohibitively costly to fix [Ref. 19]. Generally speaking, CM applies

some rigor and controls to a process in which programmers (with their artisan-inspired drive for optimization and resulting increased costs) vie with their managers, who seek to control the software baseline and the costs that accompany any changes to that baseline. [Ref. 19]

Considering that this project is a Government in-house effort, cost control measures and concerns have taken on less significance. For the Interim AMPS, programmatic and software support is provided by omnibus-type contracts with Systems Dynamics International, and Vitrinics, Inc., respectively. Since system integration is performed by C2SID, CECOM (a Government agency), no contract is required for this effort. In addition, there seems to be less emphasis on the level of management oversight that would exist if a commercial contractor were performing all the development/integration work for the Government. [Ref. 9]

c.   *Testing/IV&V*

Software testing, an umbrella term to categorize those activities carried out to confirm the presence of software defects [Ref. 8], is present within the Interim AMPS development process [Ref. 32]. Programmers conduct software unit and component testing through what is termed "lower-level" or "desk-top" testing [Ref. 32]. Distinct software module testing is conducted by both the individual programmer and the software engineer/system integrator. The process is informal, and does not emphasize documentation. While no formal procedures are in place, the above activities serve to validate software units, components, and modules prior to them

progressing into the CM system [Ref. 32]. Successfully tested and validated modules are then passed into the configuration management system and subsequently to SED-AV for formal testing. For the Interim AMPS, "formal" testing focuses on software defect detection and system-level validation. Here, testers essentially attempt to "break" the software. Qualification testing is not conducted, because of the rapid-prototype nature of the software and because only the system-level requirements for the software have been identified. When all software requirements (i.e., down to unit level) have been derived, formal qualification testing will be possible. Essentially, this is the future of SED-AV testing for the production AMPS software. [Ref. 32]

Independent Verification and Validation tasks are currently being conducted for the Interim AMPS by SED-AV, CECOM [Ref. 32]. This includes testing the performance of the software, and determining that it satisfies all system-level requirements. However, because of the absence of a completed Software Requirements Specification (SRS), systems engineering analytical activities are restricted to the system as a whole. Therefore, true IV&V activities are not possible within the Interim AMPS process [Ref. 32]. An expanded discussion of IV&V process attributes will be addressed in the examination of the development process for the AMPS.

d. Management Principles

It is not appropriate to measure the Interim AMPS software development process against a guideline for management principles. The Interim

AMPS, was originally intended as a proof of principle [Ref. 3], technology insertion, and risk reduction vehicle [Ref. 24]. As such, the program represents a tool available to the software development manager [Ref. 6]. In this regard, it is among the most powerful tools available to analyze and refine requirements [Ref. 6].

The status of the Interim AMPS as a prototype gives more license to developers and managers. It calls for minimal constraints on choice of programming languages, documentation, and the use of standards [Ref. 6]. Typically, the prototype/rapid prototype methodology entails near unconstrained development of a functional software package.

The Interim AMPS has recently undergone a transformation. Fiscal constraints have mandated the change in the nature of the Interim AMPS program, from prototyping to production effort [Ref. 25]. The challenge for both developers and managers is to convert their organizations and processes from software prototyping orientation to production software orientation. In this regard, the wheels are already in motion. Some background and the reasoning behind the paradigm shift from prototype to production system is now presented.

B.   "AMPS" : THE PRODUCTION SYSTEM

The original acquisition strategy for the AMPS called for the production version of the software to be contracted to a commercial software development entity [Refs. 25, 31]. In February 1994, an AMPS Process Action Team (PAT) was established with the purpose of "laying out a program that would develop a single

mission planning system that meets the requirements of the entire fleet" [Ref 27].

The PAT was to leverage existing mission planning efforts and use Non-Developmental Item hardware and Commercial-Off-The-Shelf software solutions whenever possible [Ref. 27].

The AMPS PAT considered the five courses of action (COAs) shown in Figure 10 [Ref. 27]. After careful evaluation, a PEO Aviation development effort was proposed. This would use the Lockheed Sanders' Air Force Mission Support System (AFMSS) as the core capability (COA #1.) AFMSS was a proven aviation mission planning software shell which allowed for the integration of aircraft specific software modules into the basic shell [Ref. 27].

COA #1 was thought to be superior on budgetary (it was cheaper than a new, unique Army system), political (OSD did not want any new mission planning developments), interoperability (e.g. with the Air Force), and technical (e.g. demonstrated capabilities) grounds [Ref. 27].

The other COA of interest to this research effort, COA #4 - Use of the CECOM developed Interim AMPS as the core software, was determined not viable for the following reasons:

(a) Due to the rapid fashion of the feasibility effort, adequate software documentation was not performed. This lack of documentation included both 2167A management documentation (e.g., SRS, SDD, etc.), and technical documentation (e.g.,

Number	Course of Action
1	Use the Air Force Mission Support System (AFMSS) as the core software.
2	Use the Navy's Tactical Aircraft Mission Planning System (TAMPS) as the core software.
3	New Start Program- Development of an ORD compliant system from scratch.
4	Use CECOM developed Interim AMPS as the core software.
5	Use Comanche contractor developed software as a basis for a single system.

Figure 10. The AMPS Process Action Team Courses of Action [Ref. 27].

software coding comments.) The PAT therefore questioned the ability to maintain the software. [Ref. 27]

(b) A rapid prototyping approach was used which focused only on producing functional code. The resulting code was not modular and it did not use a top down structural approach. Therefore, much of the resulting structure was unstructured or spaghetti-like in nature. [Ref. 27]

The PAT concluded that these factors made post-deployment software maintainability and supportability questionable. Because of the unstructured nature of much of the code, expanded functionality would also be difficult, if not impossible. To resolve these problems the code would have to be redesigned, and documented in accordance with DOD-STD-2167A. Since this task would take an estimated two years

to accomplish, the approach was deemed only marginally better than a new start program. [Ref. 27]

Fiscal reality, however, was to play a pivotal role in the COA chosen. Though initial discussions with Lockheed Sanders, Inc. had suggested the AFMSS approach would be "affordable" (approximately $5 million) [Ref. 31], the subsequent proposal from the company was approximately $20 million over the budgeted Program Objective Memorandum (POM) amount [Ref. 32]. PM-AEC simply could not afford Lockheed Sanders, Inc. to be the system developer/integrator for the AMPS [Ref. 32].

The "Revised Strategy," depicted in Figure 11 represents both an acknowledgement of budgetary constraints and a large assumption of risk. The Interim AMPS, per se, is no more. The software prototyping approach has been replaced by a production-software development approach [Ref. 25]. The effort will be conducted in-house by C2SID, CECOM. SED-AV, CECOM will provide formal qualification testing, IV&V, certification, and eventually PDSS for the AMPS. [Ref. 32]

The following section is an overview of the production-software development process for the AMPS. The areas of focus are the same as those examined for the Interim AMPS prototyping process: coding, documentation, configuration management policies and procedures, test and evaluation and IV&V, and applied management principles. The intent is to highlight process change and improvement initiatives, as the development team transforms its organization and process from one adept at prototyping, to one capable of developing quality, maintainable, and supportable software.

46
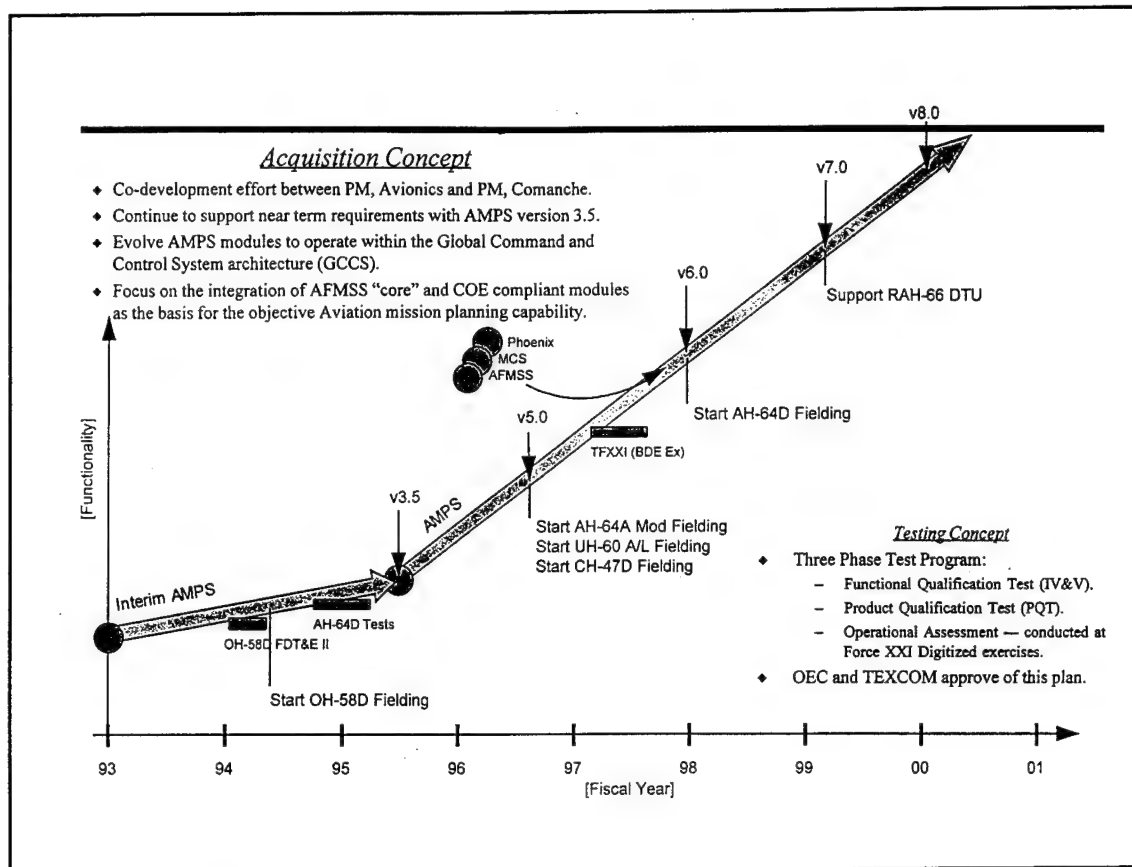
Figure 11. The Revised AMPS Acquisition Strategy [Ref. 25].

1. The Production-Software Development Process

PM-AEC and C2SID readily acknowledge a marked shift in focus within the AMPS development effort [Ref. 31]. One observer had the following insight to offer on the evolving nature of the AMPS development process/effort [Ref. 31]:

> Consider the status of our process in these terms. A married couple is expecting their first child. Typically, they have about nine months to prepare the infant's nursery. The AMPS development team is akin to a couple that **was not** expecting a child, and one day took delivery of a one-year old.

47

Accompanying the arrival of "the child" is the realization that organizational and process change is inevitable and desirable [Refs. 25, 31]. How close the team can come to a process optimized for the development of production software, however, is a matter of debate [Ref. 31]. The goal is the production of reliable, maintainable, and supportable software. One method for achieving this is through the application of a structured discipline imposed by a software engineering process [Ref. 8]. Examples of some software engineering practices that can be applied to any software development effort include [Ref 8]:

- Quality engineering,

- A formalized software development process,

- Informal/formal peer inspections,

- Rigorous configuration management,

- Continuous process improvement,

- Statistical process control,

- Defect causal analysis and prevention,

- Quality monitoring metrics and interpretation,

- Employment of Integrated Process and Product Development Teams.

This environment ensures reliability, maintainability, and supportability are designed into the system, rather than retro-fitted in after deployment [Ref. 8]. A commitment to software engineering forces a movement away from the "build-it-quick, get-it-to-the-field" mind-set. Instead, resources are planned and managed within a total life-cycle framework. [Ref. 8]

48

The AMPS management and development personnel are acutely aware that process change/improvement is necessary to reach their goal of fielding quality software. Process improvement initiatives are under consideration (or underway) in several areas of the development process [Refs. 25, 31]. The chapter now turns to a discussion of some of those initiatives in the same areas looked at for the Interim AMPS: coding, documentation, configuration management policies and procedures, test and evaluation and IV&V, and the application of management principles.

a. *Coding/Technical Documentation*

Both formal and informal initiatives are underway to make the code more maintainable and supportable. They encompass an informal policy to further "break out" coding responsibilities to programmers, and an increased emphasis on the transfer of code-design logic and programmer intent. In addition, there is an increased emphasis on code commenting (technical documentation). [Ref. 31]

The practices employed for the Interim AMPS, make it questionable whether the code could be maintained or supported by anyone other than the original programmer. The emphasis on producing functional code was accomplished at the expense of maintainability and supportability [Ref. 31]. For instance, one programmer estimated that only about five to ten percent of his code for the Interim AMPS was commented [Ref. 32]. Additionally, the unstructured nature (i.e., other than top-down) of the design has resulted in "spaghetti-like" code for an estimated ten to twenty percent of the program. [Ref. 32] Much of this code now forms the basis

49

for the production variant of the AMPS, driving the need to correct these weaknesses. One approach chosen by the development team is a reverse-engineering effort. [Ref. 32]

This formal method employs a computer aided software engineering (CASE) tool. The desired output of the tool is a comprehensive software design document (SDD), a data flow diagram. The goal is to identify the "spaghetti code," replacing it with workable, structured code/modules. In addition, the chosen tool will provide an on-line documentation and maintenance function. This will allow documentation to be conducted "on the fly" by programmers, eliminating the need for further after-the-fact technical documentation. [Ref. 32]

Some of the CASE tools being considered/tested are "Cadre," "Ensemble," "Teamwork," and "Hindsight." All have particular strengths and weaknesses. Unfortunately, no one tool has consistently demonstrated all desired attributes. As this research was conducted, the "optimal" tool had yet to be determined. All agree however, that the employment of a CASE tool is necessary and will enhance the ability of programmers to maintain and add modules. Additionally, the streamlining and increased effectiveness of the documentation process is an added benefit of the tool. [Ref. 32]

b. Configuration Management

With the AMPS software development changing from a prototyping to production effort, the need for adherence to a more formal/rigid configuration

50

management system is recognized [Ref. 32]. The primary reasons for configuration management (identification, communication, and cost control) assume greater importance as the scope and complexity of the development effort grow. Figure 12 graphically depicts the current AMPS software schedule [Ref. 25]. Note the distinction between annual major releases (e.g., major functional enhancements), semi-annual minor releases (e.g., minor functional enhancements), and unscheduled maintenance releases (e.g., correction of bugs) [Ref. 25]. The AMPS Capabilities Matrix, not depicted, details the added functionality of each release [Ref. 30].

The broadened scope and complexity of the AMPS development, and the necessity to track the software baseline, has resulted in increased emphasis on the adherence to the established configuration management process [Refs. 25, 31]. This process is simple and effective, if employed appropriately. A discussion of the configuration management process follows.

The process begins with the configuration manager (CM) submitting a System Change Request (SCR). These are based on System/Software Trouble Reports (SSTRs) generated by user, developer and testing activities. Early analysis of the SCR is performed to determine the type of SCR (e.g., fix, non-problem such as operator error, or enhancement); estimate of schedule, cost, manpower, and technical impacts (e.g., files affected, rippling effects, system performance, etc.); and recommended priority. [Ref. 23]

51

	FY 94	FY 95	FY 96	FY 97	FY 98	FY 99	FY 00	FY 01	FY 02	FY 03
In-Process Reviews		IPR ◆			◇ IPR					
Development			Version5.0 5.1 5.2							
			Version 6.0							
				Version 7.0 7.1						
				Version 8.0						
OT&E				□ Task Force XXI						
Deliveries			OH-58D KW							
			UH-60A/L							
			AH-64A Mod							
			CH-47D					CH-47D (128B)		
				AH-64D LB						
						RAH-66 (Prototype)			RAH-66	
	FY 94	FY 95	FY 96	FY 97	FY 98	FY 99	FY 00	FY 01	FY 02	FY 03

◆ Annual Major Releases (Version 5.0, Version 6.0, etc) Consisting of Major Functional Enhancements
◆ Semi-Annual Minor Releases (Version 5.1, Version 5.2, etc) Consisting of Minor Functional Enhancements
◆ Unscheduled Maintenance Releases (Version 5.1a, Version 5.1b, etc) to Correct Bugs

Figure 12. The AMPS Software Schedule [Ref. 25].

The Configuration Control Board (CCB), composed of development, management and user personnel, receives the SCRs with the early analysis information and either approves, defers, or closes the SCRs. The CM then updates the SCR tracking database, reflecting CCB decisions, and the SCR information source is notified of the SCR disposition. [Ref. 23]

Open/approved SCRs pertaining to software changes are implemented and tested by the AMPS project development team and a new baseline (i.e., source and executable code, and software development tools) is complied. The executable code baseline is subsequently released for formal testing (i.e., Formal Qualification Testing.) SSTRs generated from formal testing or SCRs for which formal testing

show have not yet been correctly implemented will be cycled back into the process at the appropriate step, until closed. Finally, the SCM releases the software for replication/distribution upon authorization. A diagram of the process appears in Figure 13. [Ref. 23]

The proper and consistent application of this process ensures that tenants of configuration management practice have been met. The "current" software version is formally identified to all parties when the SCM releases the software for replication and distribution after authorization. This allows effective communication about the software between all concerned entities (e.g., management, developer, user, etc.) This is true because all parties are able to maintain a mutual understanding of changes made to the software (e.g., fixes, functional enhancements, etc.) Cost control is facilitated by the CM process through the use of the CCB in its role of SCR reviewer. Here, fixes and enhancements that are not cost-effective are quickly ruled out or deferred until resources are available to address them.

Observed management-driven emphasis on adherence to configuration management procedures is a good indicator that CM tenants will be met within the AMPS process [Ref. 31]. For effective configuration management, clearly, there must be firm commitment from "the top." The full challenge is to ensure that management has instituted and enforces an appropriate level of CM effort, while developers fully understand and embrace the importance of these practices as well. [Ref. 19]

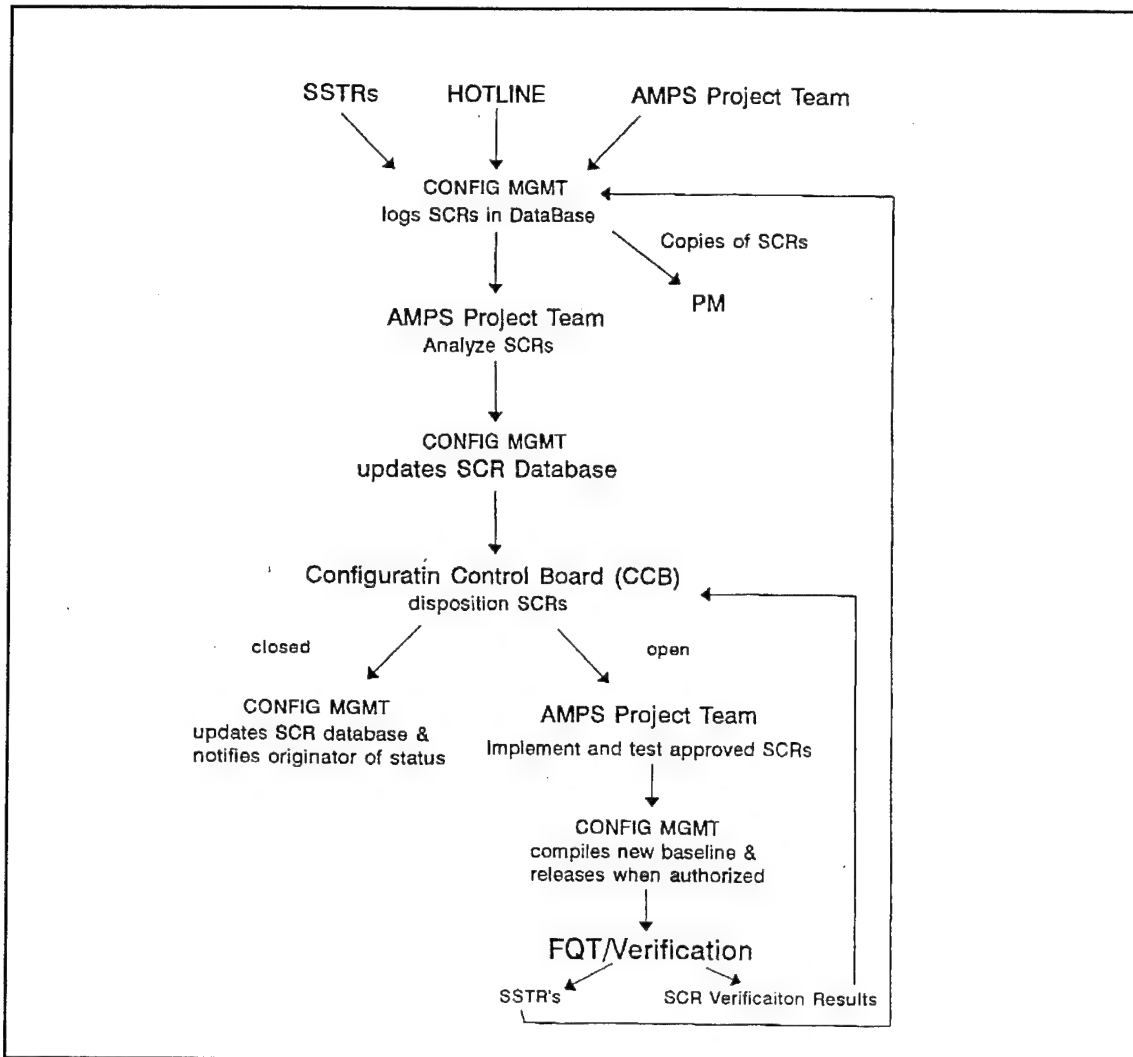Figure 13. The AMPS Configuration Management Process [Ref. 23].

c.   *Testing/IV&V*

In addition to "lower-level" and "desk-top" testing that occurred with the Interim AMPS [Ref. 32], the testing process for the AMPS employs formal software qualification testing and a documented set of test procedures. Early on, SED-AV, CECOM will focus its energies on the generation of the Software Requirements

Specification (SRS) and Software Test Description (STD). This effort is to begin 1 September 1995. Seven months have been allocated for SRS and STD generation [Ref. 36]. The intent is to generate a complete requirements list that can be traced to code and vice versa [Ref. 32]. Until this is complete, SED-AV will rely on the Top Level System Requirements List (TLSRD, 8 June 1995) for testing at the system requirements level [Refs. 36, 29]. Figure 14 shows SED-AV's proposed testing approach up through version 5.0 [Ref. 36]. By version 6.0, it is anticipated that all requirements (i.e., system-level and derived) will be known [Ref. 32].

According to the Draft Test and Evaluation Master Plan (TEMP, 29 July 1995) [Ref. 28], the developer (C2SID) conducts software testing up to CSCI-level at his facility. It should be mentioned that in its present configuration, there is only one CSCI for the AMPS. In addition, a Formal Qualification Test (FQT) for each version will be conducted by the developer, and witnessed by SED-AV, CECOM and the Test and Evaluation Command (TECOM.) The FQT includes flight performance certification by the Aviation and Troop Command (ATCOM), Directorate of Engineering (DE), and a safety assessment by the ATCOM Safety Office. [Ref. 28]

In addition, a Production Qualification Test (PQT) will be conducted by the Army Technical Test Center (ATTC). The PQT is intended to serve as a limited operational assessment. As functionality for an airframe is added to a release, the PQT will include an assessment of the AMPS compatibility with the aircraft. Prior to distribution of the software, PM-AEC will conduct a Special In Process Review (IPR)
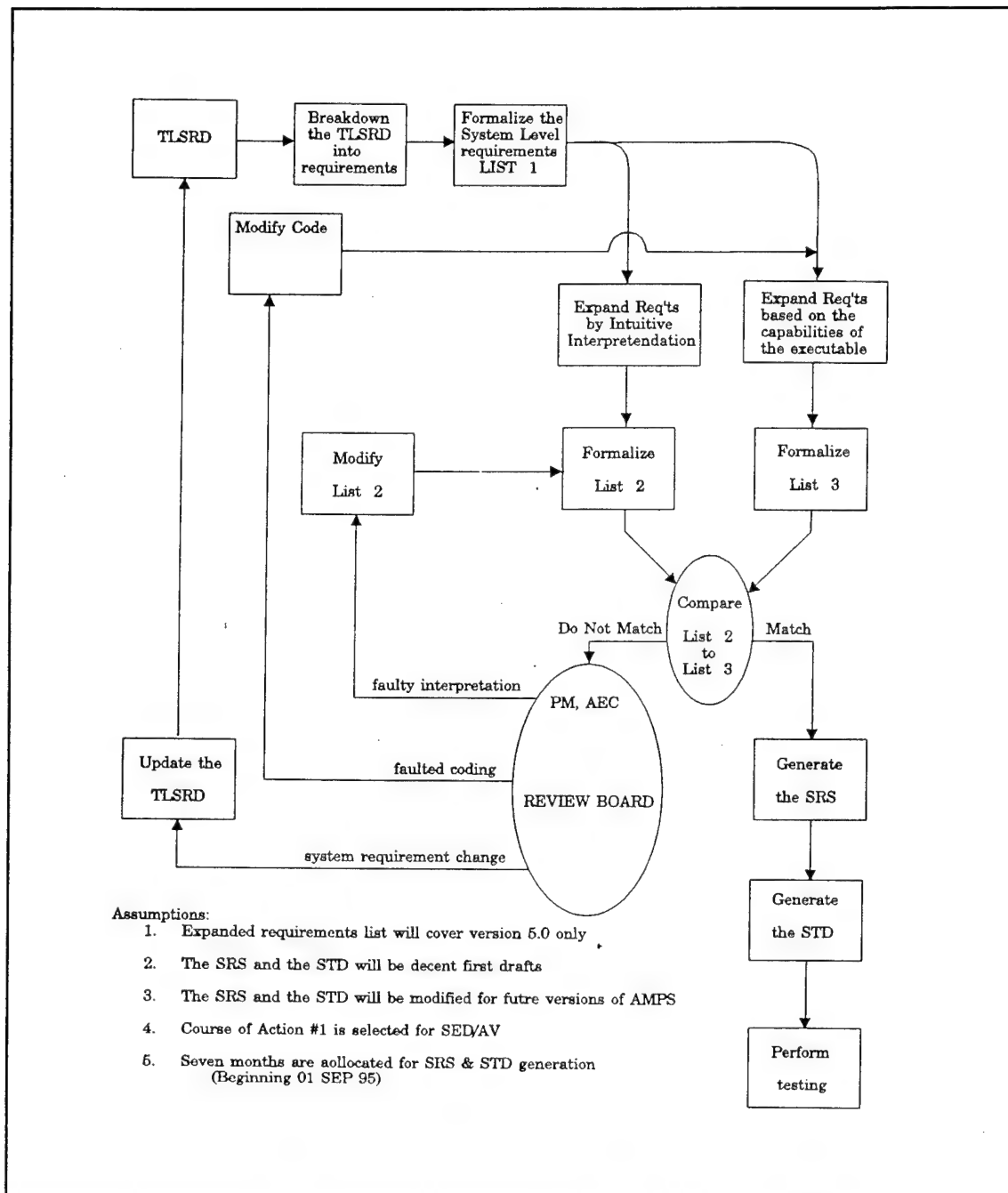
Figure 14. The SED-AV Testing Approach for Version 5.0 [Ref. 36].

with the specific airframe PM, Training and Doctrine Command (TRADOC) System

Manager (TSM) for the target aircraft. [Ref. 28]

Prior to release to receiving units, the New Equipment Training Team

(NETT) will conduct an acceptance test for each AMPS. This will be the case for all

AMPS delivered to operational units before Milestone (MS) III approval. The

Operational Assessment (OA) and the MS III decision point, is the "Force XXI-

Brigade 97" digitized exercise. This is scheduled for FY 97 at Fort Irwin, California

[Ref. 25]. The AMPS, version 5.2 will be evaluated during this OA.

For the AMPS, it has been determined that IV&V will be carried out by

SED-AV. The level of the effort in this area, however, has yet to be determined. In

addition, the "independent" aspect of IV&V is being somewhat subjugated, as SED-

AV is heavily influenced by the developer in terms of funding and direction. [Ref.

36]

It is desirable to select an IV&V agent from within the prime

development contractor (e.g. C2SID). It is also advantageous to use the software

support activity (e.g., SED-AV) in this role. However, the autonomy of the IV&V

agent is of paramount importance [Ref. 6]. SED-AV has raised this point and

proposes the use of PM-AEC as a "referee" of sorts when disagreements arise

between SED-AV and C2SID [Ref. 32].

In addition, until completion of the Software Requirements Specification

(SRS), which should reflect the requirements allocated from the System/Segment

Specification (SSS), it will be impossible to carry out the "verification" portion of

IV&V [Refs. 6, 36]. Verification, which is Computer Software Configuration Item (CSCI) oriented, evaluates how the SRS supports the SSS, and how the CSCI design supports the SRS as the design progresses to greater levels of detail [Ref. 6]. SED-AV stipulates that with the anticipated completion of the SRS for version 6.0, the nature of the IV&V will become more traditional [Ref. 32].

Validation, which is system oriented, comprises evaluation, integration, and test activities accomplished at system level. It ensures that the system satisfies the requirements of the System/Segment Specification [Ref. 6]. Validation of the system is a more accurate description of what SED-AV plans to accomplish prior to version 6.0. Previous software versions will be tested against the TLSRD [Ref. 36]. This "top-down" method of testing has the advantage of repeatedly testing top-level modules as more and more lower-level modules are coded, integrated, and tested [Ref. 6].

Certification is the ultimate goal of the IV&V effort. The term refers to the using command's agreement that the acquired system satisfies its intended operational mission [Ref. 6]. SED-AV plans to certify version 6.0. This will occur after the Operational Assessment/Test (e.g. Force XXI- "Brigade 97" Digitized Exercise, FY 97) if the software has been deemed suitable, supportable, and operationally effective [Refs. 6, 36].

d.   Management Principles

As the AMPS software transitions to a production version, both management (PM-AEC) and developer (C2SID) have begun to apply a set a management principles to their processes. As for the PM, this is reflected in PM-AEC's ability to make decisions based on a "system" perspective, not allowing either hardware or software to exclusively drive decisions [Ref. 6]. Also, with the change of the AMPS status from prototype to production system, PM-AEC has incorporated a comprehensive quarterly review process. This provides a forum for integrating the system development. [Refs. 6, 26]

Additionally, PM-AEC and C2SID have selected an innovative development plan. The process is described as both evolutionary and incremental [Ref. 9]. It is evolutionary, in that the AMPS has been an operational product all along, and subsequent releases have been further refined versions of that product. It is incremental, in that, subsequent software versions will incorporate additional functional capabilities. [Refs. 8, 30]

Prototype versions in the field allow the development/management team to continue to reap the benefits of the rapid-prototype methodology (e.g., requirements definition, user feedback, and buy-in) [Ref. 8]. This vehicle for risk reduction and technology insertion dramatically increases the probability of fielding a capable, high quality, user accepted system [Ref. 24].

The use of an IV&V agent for the AMPS is a clear indicator of commitment to process improvement [Ref. 6]. Other signs of software engineering

59

application include: 1) employment of a CASE tool to assist with the code reverse-engineering effort, 2) creation of a product-oriented Work Breakdown Structure (WBS) for financial control, and 3) generation of a Capabilities Matrix, showing how each particular system-level requirement is satisfied by a particular release/module [Refs. 29, 30]. The planned, future application of a set of core metrics [Ref. 25], is evidence that statistical measurement will be applied to the development process. This is a requirement for sound engineering practices [Ref. 8].

C.   SUMMARY

This chapter examined the development processes for both the Interim AMPS and the AMPS. The process employed for the Interim AMPS was one well-suited for prototyping efforts. However, it is not optimal for the development of production software. The current AMPS process is one in transition. The development team continues to rapidly develop prototype versions of the software, yet is moving toward a process more suited for the development of quality, maintainable, and supportable software. The intent of this chapter was to distinguish between the two types of processes, and to highlight process improvement initiatives being undertaken by the AMPS development team.

Chapter IV discusses several areas of interest that the AMPS development team should consider as they continue their transition from a prototyping to a production effort. In addition, the researcher presents several items of concern surrounding current AMPS process improvement initiatives.

IV. THE AMPS SOFTWARE DEVELOPMENT PROCESS: SOME PROCESS IMPROVEMENT CONSIDERATIONS AND SELECTED ITEMS OF CONCERN

This chapter discusses some possible issues that the AMPS team (i.e., management and developer) might choose to consider as they shift their efforts from those of software prototyping to those necessary for the development of a quality, maintainable, and supportable software product. Additionally, the researcher discusses some items of concern and areas where potential problems exist as the AMPS team makes this transition.

A. TRANSITIONING THE PROCESS: SOME CONSIDERATIONS

The following quote, taken directly from the Department of The Air Force's "Guidelines for Successful Acquisition and Management of Software Intensive Systems," describes both the investment necessary and the benefits realized when an organization commits itself to process improvement/change [Ref. 8]:

> **Transitioning** a software development program into a mature, software production requires sound management practices, an unremitting obsession for process improvement, and a wise use of technology. Elevating your programs productivity is neither simple nor cheap, but well worth the investment.

The following sections forward some topics of interest to be considered by both management and developer of any DoD software program, and specifically the AMPS program, as it transitions from prototype to production software development.

61

1.   Employment of a Software Capability Evaluation (SCE)

The AMPS software development process employed by C2SID, until recently, was purely a prototype process. Now that the AMPS software development has transitioned to production software, the development process will need to transition as well. A Software Capability Evaluation (SCE) or similarly structured internal assessment would be an effective way of determining the status (maturity-level) of the present process employed by C2SID. [Refs. 8, 5]

The SCE is based on the SEI's Capability Maturity Model (CMM), which provides a benchmark of sound, proven principles for quality. It is recognized by both engineering and manufacturing and has been demonstrated to be accurate and effective for software. The purpose of the model is to allow organizations to determine their present software development capabilities and identify areas where they need improvement. The CMM characterizes process maturity based on the extent to which repeatable and measurable software engineering and management practices are performed within the organization. The SEI Capability Maturity Model is depicted in Figure 15. [Refs. 8, 21]

The SCE is typically performed by source selection teams on commercial contractors. However, the same benefits could be realized by PM-AEC and C2SID if an assessment team were to conduct an evaluation. While the "award" to C2SID has already been made, an evaluation would serve both the management and the developer

in letting each know where the development process currently stands. Risk

management strategies would stand to benefit greatly, and future initiatives for process

improvement would be clearly indicated.

MATURITY LEVEL	CHARACTERISTICS	KEY CHALLENGES	RESULTS
5 OPTIMIZING	- Improvement fed back into process - Automated tools used to identify weakest process elements - Numerical evidence used to apply technology to critical tasks - Rigorous defect-causal analysis and defect prevention	- Still human-intensive process - Maintain organization at optimizing level	P R O D U C T I V I T Y Q U A L I T Y
4 MANAGED	(Quantitative) - Measured process - Minimum set of quality and productivity measurements - Process data stored, analyzed, and maintained	- Changing technology - Problem analysis - Problem prevention	
3 DEFINED	(Qualitative) - Process defined and institutionalized - Software Engineering Process Group leads process improvement	- Process measurement - Process analysis - Quantitative quality plans	
2 REPEATABLE	(Intuitive) - Process dependent on individuals - Basic project controls established - Strength in doing similar work, but new challenges present major risk - Orderly framework for improvement lacking	- Training - Technical practices (reviews, testing) - Process focus (standards, process groups)	R I S K
1 INITIAL	(Ad hoc/chaotic process) - No formal procedures, cost estimates, project plans - No management mechanism to ensure procedures are followed - Tools not well integrated; change control is lax - Senior management does not understand key issues	- Project management - Project planning - Configuration management - Software quality assurance	

Figure 15. The SEI Capability Maturity Model [Ref. 8, 21].

2.   Improving the Software Development Process: Three Areas of Emphasis

Though certainly not an exhaustive list, the areas of process risk management,

measurement, and error/defect detection, removal, and prevention, are certainly of

paramount consideration in the improvement of any software development process

[Ref. 8]. The AMPS software development process is no exception, and could benefit

from increased emphasis in these areas.

a.   *Risk Management*

Successful management of any software intensive system is dependent on the effective use of risk identification, assessment, reduction, and control techniques [Ref. 8]. Effective risk management can help build better software at reduced cost with a relatively low investment. Productivity gains of 50% or more, and a greater probability of producing a quality product, can be realized by incorporating disciplined engineering risk analysis and management techniques into the management process [Ref. 4].

The Department of The Air Force's "Guidelines for Successful Acquisition of Software Intensive Systems" [Ref. 8], divides software risks into those associated with the software development process, and those associated with the product itself. The guidebook goes on to list the characteristics of software development that make it prone to risk . That self-explanatory list follows [Ref. 8]:

- Software developments are very complex;

- Problem element relationships can be multidimensional;

- Software problem elements are unstable and changeable;

- The development process is dynamic;

- People are an essential development element and a problem source.

Because software development is a unique, complex, dynamic, people-intensive endeavor, inherent risks accompany the process. Because of the very nature of the attributes that bring them about, the risks cannot be completely "eliminated" by any

64

level of sound management. However a proactive management approach is the most effective way to "control" them. The focus must be on identifying, assessing, reducing, and controlling associated risks. The methods employed must be at once systematic, repeatable, and based on proven principles. [Ref. 8]

The Software Engineering Institute (SEI) has developed a Software Risk Evaluation (SRE), an assessment that focuses on the management-developer relationship. For an SRE, the PM directs an independent SRE team to conduct a risk evaluation of the developer's target software development task. Typically, the focus of the evaluation is a "Top-Ten-List" of risk items that could potentially jeopardize the program's quality, cost, or schedule goals. This Top-Ten List is compiled at least monthly by the PM or an appointed Risk Advisory Group (RAG). It is completely independent of the developer's Risk Management Plan. [Ref. 18]

The manager and developer both benefit from this disinterested, objective, examination of the development process and/or software product. The examination can further be used as a benchmark against which to measure the developer's Risk Management Plan. In addition, the SCE may highlight new or innovative risk management techniques and methods. [Ref. 18]

The product of the SRE is a set of findings that are processed to provide results back to the PM. The SRE method, is depicted graphically in Figure 16 [Ref. 8]. The method could be easily applied to the development effort for the AMPS.
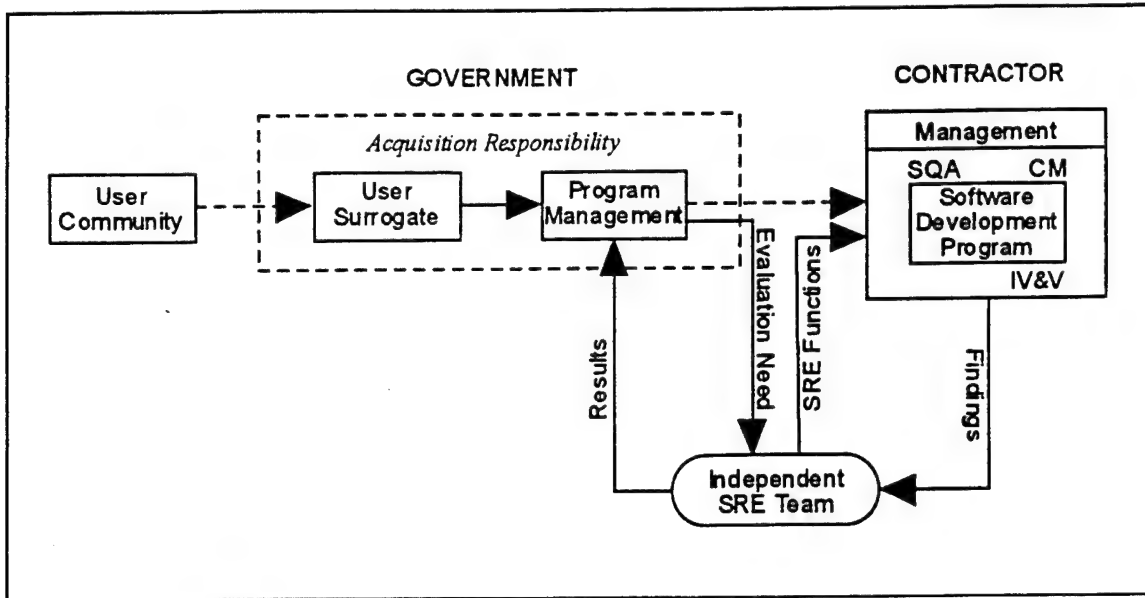
Figure 16. SRE Method Application [Ref. 8].

b.   *Measurement*

The software measurement process must be an orderly, objective method

for quantifying, evaluating, adjusting, and ultimately improving the software

development process. It is used to assess product quality, progress, and performance

throughout all software life-cycle phases. The key elements of an effective

measurement process are [Ref. 8]:

- Clearly defined software development issues, concerns, questions;

- Processing of collected data;

- Analysis of indicators;

- Implementation of process improvements.

66

Data elements are collected based on known, and anticipated development issues, concerns, and questions. These data are processed into graphical or tabular reports to aid in the issues/concerns/questions analysis. These reports and/or graphs (also called indicators) are analyzed to provide insight into developmental issues. Finally, the analysis results are used to implement process improvements and identify new issues and problems. [Ref. 8]

When employing metrics the manager/developer must ensure that the metrics are; understandable and economical (i.e., cause little extra work to generate), field tested, highly leveraged, timely and evenly spaced, and useful at multiple levels. Recent surveys conducted by the Air Force have indicated that key measures (e.g., scrap/rework) are often not collected. To correct this, the Software Engineering Institute (SEI), Government, industry, and academia have developed a set of "core metrics" that if properly employed, can be used by program managers to make informed decisions throughout the software acquisition life-cycle. These core metrics are **size, effort, schedule, quality, and scrap/rework** [Ref. 8]. A brief discussion of each follows.

Size. The management/developer should track actual software size against original estimates, incrementally, and for the total build. Data requirements for these measures include distinct functional requirements in the SRS, the number of software units contained in the SDP or SDD, and source lines of code (SLOC) or function point (FP) estimates for each CSCI compared to the actual SLOC or FP listing for each software unit. [Ref. 8]

Effort. Actual versus planned staff-hours expended should be tracked from day one of the project. It is desirable to break these labor/support staff expenditures down further into task areas, such as experience level and task assignment. [Ref. 8]

Schedule. This measure tracks performance toward meeting commitments, milestones, and dates. Entry and exit criteria for each event or milestone must be agreed upon at the outset. Only then will what constitutes progress slippage and revision be placed on common ground between management and developer. [Ref. 8]

Quality. This is a simple measure of defects in the code. Defects must be identified, tracked, and resolved, subject to rigorous configuration management rules. The defect discovery and resolution rate is an excellent measure of software health. [Ref. 8]

Scrap/rework. This is a measurement of the amount of effort lost when portions of the program must be either scrapped or reworked due to defects or performance shortfalls. More than any other metric, scrap/rework measures reveal a developer's process maturity level. [Ref. 8]

In the case of the AMPS, a set of core metrics like these would be of great use to both management and developer. Little investment would be required, and the payoffs could potentially be great.

c. *Error/Defect Detection, Removal, and Prevention*

Defect analysis is probably the most important aspect of software process improvement. If quality software is the goal, defects and their causes must be detected, eliminated, and prevented. The number of errors and defects (i.e., in the code itself) injected into the software by requirements analysts, designers and programmers, can be quite large. One researcher estimated defects per SLOC at 50-95 per thousand lines of code (KLOC.) [Ref. 14]

Often the defect cannot be detected through tests and does not show itself as output. The problem arises when the software is stressed beyond the limits of its developmental testing. It is at these times, when the software is stressed to its maximum performance, that defects in the code can become extremely costly, sometimes deadly. [Ref. 8]

Since defect free software is presently a near impossibility, the best way to proceed is to learn from our mistakes and build it right the next time. **Defect Causal Analysis** is an effective method employed in this endeavor. The object of defect causal analysis is to both discover defects, and to pinpoint what caused the defects to occur. Thus it is an effective technique for both identifying problems and preventing defects. The defect causal analysis method is driven through either the actions of a process action team (PAT) and/or peer inspection teams. These teams are made up of small groups of developers and software verifiers who analyze defects and determine their causes. These teams are also responsible for determining how to

69

remove the cause of the defect and for implementing process change. Thus the developers drive the improvement process. [Ref. 8]

Defect removal efficiency is a cumulative measure (a metric) that is defined as the ratio of defects found prior to delivery of a software application to the total number of defects found throughout its development. This measurement gives the cumulative percentage of defects that have been removed by the end of each development phase. The focus is on early removal of defects, since the cost of defect removal almost doubles with each phase of development [Ref. 8]. Some defect removal strategies include reviews, audits, inspections (e.g. informal and formal) and walk-throughs.

By subjecting the developer's work to the scrutiny of peers and/or Government management, these methods can motivate higher quality work. Formal peer inspections, it has been estimated, can eliminate approximately 80% of all software defects [Ref. 2].

Software defect prevention is a clear indicator of the quality of the development process. The success of prevention efforts is directly related to the degree of process improvement accomplished throughout the development life-cycle. The general idea is to do things better up front and avoid substantial testing and inspection expense later on. This is when defects that are found are harder and far more costly to fix. Generally speaking, the higher the "maturity level" of the process, the more effective the defect prevention initiatives. [Ref. 21]

The above methods/concepts are integral to any software development process improvement effort. In the case of the AMPS, some of these concepts are being, or soon will be, implemented. If they are embraced by management and developer alike, they will go a long way toward helping the process maintain the level where the development of quality, maintainable, supportable software is the result.

3.   MIL-STD-498 : Not Required, But Useful

Though today's acquisition environment does not promote the use of military standards, if the use of a non-government (e.g., commercial or performance) standard is not acceptable or cost effective, a MilSpec or MilStd can be used. This requires an appropriate waiver from the Milestone Decision Authority. [Ref. 7]

MIL-STD-498 was specifically developed because no commercial alternative existed or was expected to be developed for several years. Issued for an interim period of two years, the standard is to be incorporated into an International Standard Organization (ISO) standard (ISO 12207), which will have a U.S. implementing standard, IEEE 1498. This will be developed jointly by DoD and industry within the next two years. [Refs. 7, 8]

MIL-STD-498 is the principal standard for all DoD software development. It provides a framework of activities and documentation suitable for all software-intensive systems, be they weapon, C^2, or management information systems. Consisting of the standard and 22 Data Item Descriptions (DIDs), the package provides a single coordinated approach to software development within DoD. Far superior to

71

its predecessors, the standard is compatible with incremental and evolutionary development models, non-hierarchical design methods and computer aided software engineering tools [Refs. 8, 33]. A discussion of the benefits realized through the use of MIL-STD-498 follows.

Designed to accommodate "Grand Design" (waterfall), "Incremental" (e.g., Pre-Planned Product Improvement), and "Evolutionary" strategies, MIL-STD-498 is written in terms of developing software in multiple "builds." The builds can be prototypes, versions possessing partial functionality, or other partial or complete versions of the software. MIL-STD-498 is replete with instructions describing how to interpret the standard's key activities for projects employing multiple builds. [Ref. 33]

MIL-STD-498 offers alternatives to formal review and audits. Often cited as distractions from "real work," formal reviews and audits result in significant additional labor (e.g., preparation of review/audit documents, etc.) and sometimes questionable added value. In their place, MIL-STD-498 calls for more frequent, informal joint (i.e., management/developer) technical and management reviews. These reviews focus on natural work products rather than specially generated documents and materials. The idea is to perpetuate open communication between management and developer with minimum waste of time, resources, and energies. [Ref. 33]

The standard has a decreased emphasis on documentation and increased compatibility with CASE tools. MIL-STD-498 activities that call for information generation do not require the developer to "prepare a particular document," but rather to "define and record" information. This allows project information to be collected in

its natural, working form, for instance, in CASE tools. Preparation of particular documents is dramatically de-emphasized. Additionally, the standard emphasizes not making work products deliverable, whatever their form, without reason. The standard still requires the work to be performed, however the product may or may not be deliverable. A point to note here is that management still has access to the work at the developer's facility. [Ref. 33]

Additionally, MIL-STD-498 requires the developer to define and apply software management indicators (metrics) to the development effort. The developer is given the latitude to determine the metrics to be used (e.g., in the SDP), and the required management reviews. [Ref. 33]

The above are but a few selected positive attributes of MIL-STD-498. The standard is perfectly compatible with the development effort for the AMPS software, and would provide a sturdy, yet flexible, framework in which to operate. Until a comparable commercial standard is available for program employment, application of MIL-STD-498 might be a beneficial approach. [Refs. 7, 33]

This chapter has until now focused on a few areas of consideration for an AMPS process improvement effort. The chapter now concludes with a brief discussion of a few items of concern noted by the researcher during his examination of the AMPS software development process.

B.   TRANSITIONING THE PROCESS : SELECTED ITEMS OF CONCERN

It is the nature of a process in transition to be rife with identified, and
sometimes unforeseen, risk. The AMPS process is no exception to this. In the course
of this research, several areas of concern emerged. The following section discusses a
few of those areas. While none will appear surprising to anyone involved with the
AMPS program, they are nonetheless worthy of a brief discussion.

1.   Cultural Change : A Difficult Process

The concept that "change is difficult" has been around for some time.   Niccolo
Machiavelli recognized the difficulty encountered when attempting to change the status
quo. In 1513 he wrote [Ref. 17]:

> There is nothing more difficult to take in hand, more perilous to
> conduct, or more uncertain of its success, than to take the lead in the
> introduction of a new order of things.

Typically, people are most comfortable when operating in a stable, familiar
environment. Any action to disrupt that environment is likely to be met with
resistance. Effective, lasting cultural change requires a top-down commitment by
management, the empowerment of development team members, and a "team spirit," to
hold the initiative together.

Cultural change requires the institutionalizing of a new way of thinking and
working for team members. This can be brought about by changes in procedures,
training of personnel, increased (process) automation, and the addition of tools [Ref.
8]. Additionally, lasting process improvement can only occur when a rigorous

74

software engineering process is applied to the human process. Improvement objectives must be clear, and attainable through process change supported by technology (e.g., tools, automation, etc.). Above all, there must be a commitment to never return to the old ways, even when the new methods at first appear to impede the process. [Ref. 8]

Management (i.e., PM and/or developer) must exhibit an unfaltering, firm commitment to the process change, while exhibiting flexibility, common sense, and technical understanding. Realistic expectations and patience are called for as well. Any process/cultural transition is bound to be a "bumpy ride" at times. Development team members must be empowered to make both incremental and revolutionary change. Problems that arise become everyone's problems, and buy-in and ownership of process change becomes manifest to the effort. Additionally, area experts are motivated to apply their knowledge to specific problem areas. They are often the source of optimal solutions. [Ref. 8]

A shared objective and a common game plan are the hallmarks of any team. Without these elements the team cannot ultimately be successful. In addition, a team spirit must be fostered that allows it to persevere in times of difficulty. The team must be capable of rallying as a group to overcome, or outflank, any obstacle. [Ref. 8]

The AMPS software development process has been very successful to date and all program objectives have been attained [Ref. 25]. This is no small feat for a program with the breadth and sophistication of the AMPS. Many will question, or even resist, any effort to tinker with a process that has proven successful. The old

adage "If it ain't broke, don't fix it!" readily comes to mind. The truth of the matter, however, is that this is very poor advice [Ref. 8]. Especially when concerning a program such as the AMPS, that has itself undergone an evolutionary change [Ref. 25].

Furthermore, in today's rapidly changing environment of software development, what worked best yesterday, might not be optimal for tomorrow. With new tools and methods constantly being made available or improved, management and developers must exhibit a nimbleness, and a propensity to exercise the adage "If it ain't broke, break it!"

2.   CASE Tool Employment for Reverse Engineering The Interim AMPS Code : Not A "Silver Bullet"

This concept is taken from an article by Dr. Fred Brooks, that a single tool or method will be the "silver bullet" that will cure software quality or productivity problems (i.e., kill the werewolf) [Ref. 13]. Surprisingly, more than 70% of U.S. software managers believe that there are tools, methods, and concepts available that will solve many serious development problems [Ref. 13]. This myth is often perpetuated by the vendors of the tools or authors of the methods themselves, that see great gain in touting their development as the cure-all for a multitude of software development woes. The bottom line, however, is that there are no "silver bullets." There is no single method, tool, or concept that, in itself, can effect large improvements in any tangible aspect of software performance (e.g., quality, productivity, etc.) [Ref. 13]. More meaningful, by far, is a multi-faceted approach

76

toward eliminating the problem. Since software problems tend to be quite diverse, improvements should occur in parallel [Ref. 13].

In the case of the AMPS, the search is on for the optimum CASE tool for the pending reverse engineering of the code. At the time of this research, that tool had yet to be determined. Furthermore the expectations of the tool's eventual actual utility in this effort were quite diverse [Ref. 32]. It was evident, however, that some people involved with the program tended to think of the CASE tool as a "silver bullet" of sorts.

Probably the most realistic assessment of the tool's future utility was described by one member of the development team. The individual recognized both the absolute necessity of the tool in this effort, and its probable limitations. While a Software Design Document (SDD) form of output, and some type of on-line documentation and maintenance capabilities are expected, other important capabilities will likely be lacking. For instance, documenting programmer "intent" throughout the code, could very well be a manual, time intensive "Sit down with the programmer and interview him," type effort. [Ref. 32]

For the AMPS software development effort, there is a lot riding on the output of the CASE tool in the reverse engineering effort. If successful, the program will clear a difficult obstacle, and the likelihood of its future success will be greatly increased. Failure will mean a large set-back in the drive to make the AMPS software maintainable and supportable. It is human nature to hope for a cure-all in a situation

like this. However, realistic expectations and risk mitigation plans must be the order of the day if the effort is to succeed.

3. Programming Language for the AMPS : Ada is the Future

The original prototype nature of the AMPS software adequately explains the choice of ANSI C as the programming language. As already discussed, the original approach called for the AMPS production software to be written (in Ada) by a commercial contractor. The transition to the in-house effort for the development of the production software, however, has significantly changed the approach. [Ref. 25]

The AMPS program will now be looked at for conversion to Ada at the end of the development cycle (e.g., Version 8.0, FY 99). This is probably optimal from a requirements definition standpoint, as requirements should be stable by this time. Additionally, the current (i.e., ANSI C) software will be available to support the upcoming Force XXI experiments and aircraft-modification fieldings [Ref. 25]. This would likely not be the case if conversion to Ada was directed today. Therefore, immediate conversion to Ada is probably not warranted nor desirable. The program would realize little value added in the near term. Also, learning and data-gathering opportunities would be lost in the upcoming digital exercises, and hard-won user credibility would likely be compromised. [Ref. 25]

This being said, the case can still be made that realistic, well thought out plans for future AMPS Ada conversion should be a near-term program objective. Fortunately, evidence of just such planning was found at SED-AV [Ref. 36]. That

78

SED-AV is contemplating this now is a good indication. If the developer does not accomplish conversion to Ada within the development life-cycle of the AMPS software, due to fiscal or schedule constraints, it may well make sense to carry out the conversion during PDSS [Ref. 8]. But a difficult question to answer is: If we cannot afford (in time or cost) to re-engineer during development, will the funding for such a major undertaking be available during PDSS? The outlook is doubtful.

In any event, DoD is committed to Ada for the foreseeable future. If the AMPS is to be successful, it will need to be maintainable. All future software support organizations will be capable of supporting Ada products. Products developed in languages other than Ada products will be more challenging to maintain and continued support for such systems is questionable. [Ref. 8]

Many of the early limitations associated with Ada (e.g., lack of validated compilers, inadequate tool support, etc.) have been mitigated or corrected [Ref. 8]. The reasons for using Ada for embedded software become more compelling every day. Its benefits are many (as depicted in Figure 17 [Ref. 8].) Ada acts as an enabling technology for a sound, engineered software development process [Ref. 8]. Today, more than ever, Ada makes sense. Especially, for systems as integral as the AMPS will one day be.
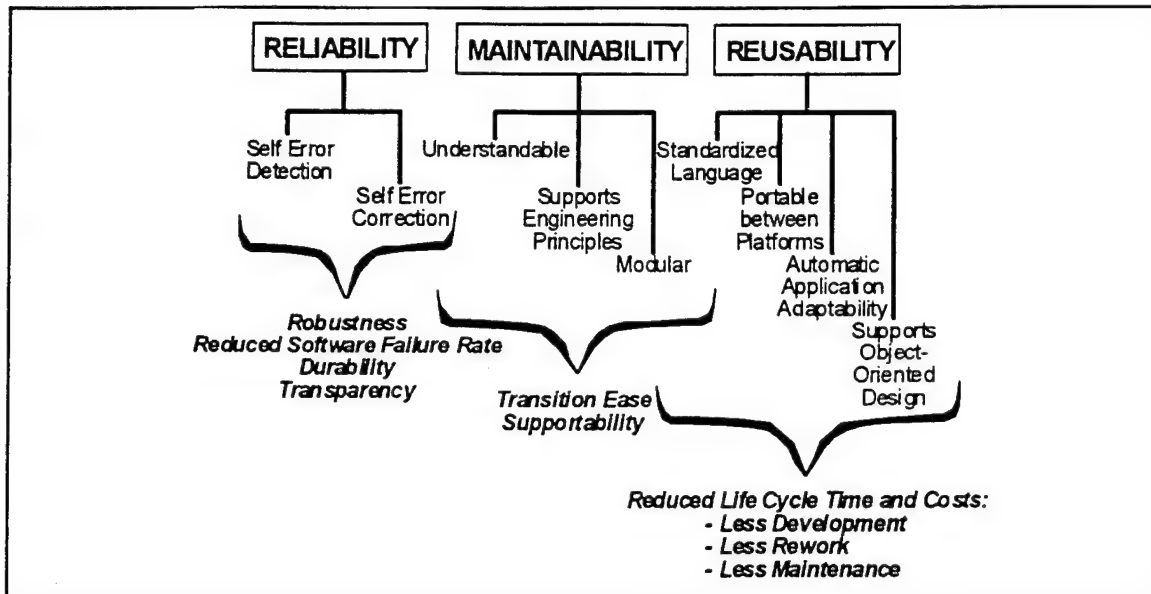
Figure 17. Summary of Ada Feature Benefits [Ref. 8].

C.   SUMMARY

This chapter offered some insights that the AMPS development team might consider as they transition their software development process to a new phase. Addressed were; 1) Selected areas to consider in the management of process improvement; 2) Possible use of an SCE to determine where the AMPS software process is, and where it needs to go, and; 3) Some of the benefits that could be realized through the us of MIL-STD-498. Additionally, a few selected items of concern were briefly discussed. They included; 1) The difficulties of implementing cultural change; 2) The employment of CASE tools and their limitations, and; 3) The benefits of Ada and the need to plan in depth now for Ada conversion in the future.

V. CONCLUSIONS AND RECOMMENDATIONS

This chapter presents the conclusions and recommendations drawn from this study of DoD software development activities, specifically those employed in the development of the AMPS. The answers to the primary and subsidiary research questions are also presented. The chapter concludes with recommendations for further research.

A. CONCLUSIONS

In the past, software development activities within DoD have been notable for their propensity for going over budget, getting far behind schedule, and over-running projected costs. Many projects have ended in outright failure after much time and untold resources have been expended toward their development. Needless to say, desired capabilities have been left unfulfilled in many instances because of this "spotty" record of performance in the crucial area of software development.

The winds of change are blowing, however. It is now clear that software will drive the weapon systems of the future. Indeed, software is on the critical path for all major weapon system developments of the future. The Army's focus on the "digital battlefield" lends further credence to the observation that software is a key part of the future for the Army and DoD.

Additionally, the large budgets of the recent past are behind us. DoD cannot afford to squander its resources; The old propensity of "throwing money at the

81

problem to make it go away" is a thing of the past. Therefore, methodologies, practices, and procedures must be well planned, tested, and proven. Best practices must be recognized and vigorously applied. Education must be emphasized and continuous.

Great strides have been made in the realm of software development. DoD, and commercial industry are well on the way to knowing what it takes to optimize the software development process. The use of prototypes for requirements analysis and specification, evolutionary and incremental development/delivery methods, and the development and application of sound software engineering practices/procedures are clear examples of large steps in the right direction. Development and application of standards that accommodate the above practices, while providing a framework for development efforts, is further evidence of this movement toward more effective software development.

The AMPS software development process is worthy of study because it is illustrative of a process with the potential of evolving into one optimized for software development. The groundwork has, in many instances, already been laid for this transition, and efforts are continuing. If a continued studied, consistent application of emerging concepts, standards, and methods is applied to this process, it will serve as a process test-bed for software development projects, both large and small. Challenges brought about by process transition, the methods employed in dealing with those challenges, and the end results of the new process (e.g., the software product), will be

clearly displayed. Newfound insights can be applied to software development

activities throughout DoD, adding to the growing data-base of knowledge in this area.

B.   RECOMMENDATIONS

The following recommendations are drawn from this study of the DoD software

development environment and, specifically, the case study of the AMPS. They are

applicable to any DoD software development effort, be it large or small. The list is

only representative of the myriad of concepts, methods, and procedures software

developers and managers must consider in their pursuit of quality, supportable,

maintainable software.

1.   Implement a Constant Process Improvement/Software Engineering Framework

Within the context of process improvement, three areas worthy of focus are:

1) risk management; 2) measurement of the process, and; 3) quality. Risk, because it

cannot be totally avoided, should be embraced by both management and developer.

Appropriate risk mitigation plans/techniques must be constantly formulated, revised,

and implemented. Measurement of the process must be undertaken, through software

capability evaluations (SCEs) and the application of a set of "core metrics." This will

help to determine where the process stands and what actions need to be taken to

further improve it. Quality, must be built in to the process, and the product, through

informal and formal testing procedures, as well as the application of IV&V.

2. Employ Appropriate Standards and Tailor to the Project

The amount of latitude given today to management and developer in this area is unprecedented. In this regard, there is potential for great good, to be done in the field of software development. The selection, use and enforcement of sound standards, like MIL-STD-498 and eventually its IEEE equivalent, will lay down a framework for an "engineered" software development environment. Additionally, management documentation and deliverable products that fail to add value to the process/product will be minimized.

3. Ensure All Stakeholders Participate in Requirements Definition and Analysis

Users, developers, testers, and maintainers must share these responsibilities and must ensure requirements are documented , implementable, and testable. Integrated Process and Product Development Teams are particularly well suited to this type of effort. Additionally, development of prototype software products, and testing and IV&V by the eventual maintainer, are methods to ensure this.

4. Use Commercial-Off-The-Shelf (COTS)/Reusable Components When Appropriate

Use COTS/reusable components when available and appropriate, but do not modify them. In addition, one should be aware of the associated data rights and incumbent software supportability implications.

5. Acquire/Use Appropriate CASE Tools

Encourage the introduction of standard CASE tools for all subprocesses (e.g., development, testing, maintenance, etc.). But only fund and train for those tools that will satisfy a defined need. Also, clearly understand both the benefits and limitations of CASE tools. Don't fall into the trap of thinking of CASE tools as "silver bullets" capable of solving a plethora of software development problems.

6. Use Ada from the beginning or Migrate at First Appropriate Opportunity

Earlier limitations of Ada have been largely mitigated and the benefits of using it are many (e.g., enhanced reliability, maintainability, and reusability). Ada is designed to incorporate the principles of software engineering, thus allowing the attainment of process software engineering goals. Lastly, DoD is committed to the use of the Ada language over the long run. Barring a waiver, all new DoD software products, and any major modifications and enhancements must be in Ada.

C. ANSWERS TO RESEARCH QUESTIONS

1. Primary Research Question.

What are the major features and supporting attributes of the developmental model employed for the Aviation Mission Planning Systems (AMPS), and how does this process compare/contrast with more traditional developmental models?

The developmental model/process employed in the development of the AMPS is one in transition. Initially, the process employed for the Interim AMPS was a

85

software prototype process. The primary objective was the production of functional code for prototype fielding and evaluation/feedback. Inherent in this process were those attributes favorable for rapid production of software prototypes.

In software prototypes, code design structure/architecture is of secondary importance to functional code, and a top-down methodology of code design development is abandoned. Additionally, management and technical documentation are typically slighted, and configuration management procedures are cursory or often bypassed. Testing/IV&V is informal or cursory. Quality software development is second in priority to rapid functional software development.

These attributes, while favorable to the development of prototype software, are unfavorable to the development of production-quality software. In light of this, in conjunction with the fact that the AMPS is now to become production/fielded software, the development process has begun a transformation.

The development process or model now in use for the development of the AMPS could be characterized as evolutionary. The AMPS software has sustained an operational product, with limited capabilities, from its initial fielding as a prototype. More refined versions, with increased capabilities, are currently being developed.

Most aspects of the process are undergoing change, modification, and improvement. Coding design structure will undergo an overhaul, beginning with the application of a CASE tool, that will produce a design document (code structure) and assist with code technical documentation efforts. Configuration management procedures are in place that will assure software identification, communication, and

cost control procedures are implemented. Testing and IV&V is also becoming more robust and will employ informal testing, formal qualification testing, and a documented set of test procedures. Certification for the AMPS software, the ultimate goal of the IV&V effort, should be possible by the first "true" production version (i.e., 6.0). Central to the effort, is the software engineering framework that is gradually being applied to the process.

The AMPS development model is an improvement over earlier, more traditional ones (for instance the "waterfall" model). The evolutionary development model allows far more flexibility on the part of management and the developer. The model is conducive to the employment of more robust and flexible standards, like MIL-STD-498. This not only streamlines the process for the development team, (e.g., reducing documentation requirements, deliverables, etc.) but provides a sound software engineering foundation for the development effort.

Additionally, unlike the waterfall model, early user involvement is inherent in this model. This is because of the requirements of this model for the development and demonstration of software "increments" (i.e., additional capabilities). Also, this strategy is particularly suited to situations where the general scope of the program is known, but only a basic core of functional characteristics can be defined or detailed system-level requirements are difficult to determine. As requirements are further defined, functionality and changes are able to be added because of the flexible, modular nature of the core capability. This does not hold true for the waterfall model and its products. Lastly, when an evolutionary strategy is employed, developmental efforts

are conducted within the confines of a plan for advancement to an end capability. With the waterfall model, the "plan" may change as requirements are added or change, and the end state or capability becomes a moving target.

2. First Subsidiary Question.

What are the primary features and attributes (both beneficial and detrimental) of traditional software development methodologies (waterfall, sequential, etc.) that were primarily used in conjunction with DOD-STD-2167A?

More traditional models of software development, like the "waterfall" model, were a good first step toward applying a disciplined software engineering framework on an environment previously characterized by a code-and-fix method of software development. This type of strategy placed emphasis on initial requirements and design activities and on producing documentation during the early developmental phases.

However, this strategy does not support modern developmental practices like prototyping and automatic code generation. Additionally, initial requirements are seldom comprehensive. They are added to, or change throughout the development process. Also, documentation requirements for this model tended to be excessive while the model's associated standards are inflexible and requirement heavy.

The waterfall-type model is "lock-step" in nature, each activity being a prerequisite for following activities. Additionally, the model does not expose integration problems until the later stages of development, when fixes are far more

difficult and expensive. Lastly, using this strategy, a finished product is not available until the end of the process. This tends to discourage user involvement.

3.   Second Subsidiary Question.

Citing recent developments in software engineering, and the directed movement away from reliance on MIL/DOD-STDs, what are the attributes of more current models employed in the development of mission critical computer resources (MCCR) for major weapon systems?

More current methods or models for software development emphasize and accommodate the application of a software engineering environment. This includes the use of CASE tools, and compatibility with software engineering methods employed in the areas of measurement, analysis, and design, as well as coding, testing, and reuse. Lastly, the models support procedures, training, and people, as they relate to the application of an engineering discipline.

Current models or strategies (i.e., evolutionary development or incremental delivery) accommodate the application of new, more "user friendly" standards (e.g., MIL-STD-498 and/or IEEE 1498) and emphasize tailoring with their use. With these, by-products of the process that are not value-added are eliminated (e.g., unneeded documentation requirements/products, etc.)

These models emphasize early user involvement and accommodate the addition of software functionality and changing requirements. Also, the models are compatible with Ada, and concepts like object oriented design (OOD). Lastly, the

models are more compatible with the way in which software is actually developed, where efforts are repetitive, often in parallel, and non-sequential.

4. Third Subsidiary Question.

What improvements are realized when MCCR is developed through a process such as the evolutionary model employed in the instance of the Aviation Mission Planning System?

The improvements realized through the employment of the evolutionary model are many and diverse. Following are but a few of the benefits realized. First, this model allows for better requirements definition, as software prototypes allow the user and developer to better define needed capabilities. This translates to a more stable development environment, as most requirements are defined up front. Regardless, the model accommodates both additional functionality, and requirement changes throughout the process.

Early user participation and feedback through the prototypes serves to facilitate buy-in, while getting a product to the field earlier than otherwise planned. Additionally, needed interfaces are more readily identified and integrated.

In addition, defects and errors are found earlier when an evolutionary process is employed. Using a model such as the waterfall process, defects and errors are often not found until later in the process (e.g., during integration activities). This translates to easier, quicker, less resource intensive fixes for the evolutionary model, and a higher quality product in the end. Hence, the product that emerges from the process is

more capable, as user defined requirements have been met, the user has accepted the product, and he is comfortable with its use. In addition, the quality level of the product is significantly higher.

D. RECOMMENDATIONS FOR FURTHER RESEARCH

1. Re-examine the AMPS Software Development Process

The process is currently one in transition and it is not clear whether process improvement/optimization initiatives will be embraced and/or successful. It would be interesting and informative to re-examine the process some time in the future to determine if process improvement initiatives have taken hold, and paid off. Along these same lines, the researcher could identify problem areas of the transition for management and developer and investigate how these challenges were overcome.

2. Cost-Benefit Analysis of AMPS Conversion to Ada

Conduct an in-depth analysis of the cost versus the benefits of converting the AMPS to the Ada programming language at some point in the life-cycle of the software. The approach could examine the feasibility of converting to Ada during the software development process and/or during PDSS. This research would represent a valuable product to future AMPS management, development, and maintenance staffs.

3. Design the "Optimized" Software Development Organization and Process

Applying today's evolving standards, methodologies, concepts, etc., design the "optimized" software development organization and process. Detail how a

development effort would be conducted within this organization and process. The intent should be to design the organization and process so they are as streamlined as possible, yet still capable of producing production-quality software. This model organization and process could be a valuable tool, serving as a benchmark for other software development activities within DoD.

QUESTIONS FOR THE AMPS DEVELOPMENT TEAM

AMPS Development Process-General

1. What would constitute an accurate description of the AMPS development process approach; evolutionary, incremental, spiral, a combination?

2. How are systems interfaces (e.g. A2C2, AVTOC, IDM, etc.) being managed so the AMPS will not be a stand-alone system within the Force XXI architecture?

3. How has the evolution in use of standards (e.g. DoD-->MIL-->Commercial) impacted the AMPS development process?

4. As Force XXI user requirement are bound to continue to evolve as needs become better understood, how is the AMPS being isolated from the harmful effects of "requirements creep?" Will this be accomplished through rapid prototype, evolutionary development/delivery, incremental design?

5. What is the organizational structure of the CECOM RDEC (AMPS) development office? Of interest are positional attributes and responsibilities, manning, experience levels, etc.

6. Generally speaking, what type of software engineering environment exists within the AMPS development office? Within the SEI's Capability Maturity Model (CMM), where does the organization fall? When was the last Software Capability Evaluation (SCE) completed?

7. How many source lines of code (SLOC) does the AMPS program currently contain? Is the effort also being measured by function point (FP) or other measurements?

8. The program is currently written in ANSI C, to Computer Software Configuration Item (CSCI) level; Are there possible plans to convert to Ada for the production version of the program?

9. Generally speaking, from requirements analysis to CSCI integration and testing, how does the AMPS developmental process compare/contrast to a typical DoD-STD-2167A driven "waterfall" process? Can you provide a general chronological description of the process?

10. The Battle Labs; Will they assist/perform system integration/test for the AMPS within the Force XXI system architecture?

AMPS Development Process-Configuration Management.

1. How are/were CSCI's ID'd/generated? Usually, these are based on the RFP/WBS, does this hold true for the AMPS?

2. Is there a published software development plan (SDP) and configuration management plan (CMP) for the program? Who is the designated configuration manager?

3. How are Class I and Class II changes controlled for the program? Does the Government employ a Configuration Control Board (CCB)?

4. What baseline has been established for AMPS? A Functional, allocated, product baseline, or an informal/developmental baseline?

5. Was the Air Force Mission Support System (AFMSS) used as a baseline? If so, how will you ensue that baseline changes/problems found "down the road" by the Air Force are communicated to your (or the PDSS) office?

6. Does CECOM RDEC or your office specifically, employ a CCB to formally process changes to the AMPS baseline?

7. Does CECOM RDEC or your office employ a S.W. configuration review board (SCRB) to review/evaluate all proposed changes to the s.w. baseline and to process/dispose of the s.w. problem reports (SPR's)? If yes, who makes up the review board?

8. Does CECOM RDEC or your office employ a s.w. development library where AMPS related data is stored for future use? Does this library also perform as the central point for configuration management?

9. Are s.w. development folders (SDF's) maintained on all AMPS CSU's, CSC's, and CSCI's?

AMPS Development Process-Test/Eval. & IV&V

1. Is there a published s.w. development test plan (SDT) for the AMPS?

2. Informal testing of CSU's and CSC's, undoubtedly is conducted routinely, but what about formal testing and the test readiness review (TRR)?

94

3. So far, has AMPS testing been of a CSCI (integration) testing nature?

4. How is/will "hot bench" testing be conducted? Will it be conducted within the current hardware configuration or the target configuration?

5. Will DT&E and OT&E be combined or sequential? The AMPS seems to lend itself to combined testing. Comments?

6. Will all three types of testing take place within the AMPS process (e.g. human, s.w. only, and integration)?

7. Will AMPS testing be bottom-up, top-down, or combined? Will test reviews/audits (type/frequency) be IAW DoD-STD-2167A, MIL-STD-498, or commercial practices?

8. How will the environment in which the AMPS will operate be simulated during hot bench/system integration testing as some of the other systems are at earlier developmental stages than the AMPS? Do system simulators exist for these other systems?

9. Is the AMPS considered a mission critical system requiring IV&V? Generally speaking, what is the IV&V approach to the system/process?

10. Was test s.w. bought/brought with the AFMSS "core" system or are you to develop your own?

11. Is the AMPS IV&V level (e.g. task level) I, II, or III?

12. Are there/will there be criteria/thresholds established for the termination of IV&V efforts?

13. If IV&V is to be employed, are there currently any good cost estimates available for this effort (percentage of total cost of AMPS development.)

14. Has the need for IV&V increased/decreased with the adoption of MIL-STD-498 and now commercial standards?

15. Has any consideration been given to the most suitable IV&V agent? Will the effort be in-house or an outside contractor?

AMPS Development Process-Applied Management Principles

1. Speaking to the industry movement to accommodate "evolving requirements," does your process do this and if so, how? Is evolutionary development/delivery being used?

2. Software development requires adequate documentation to allow for s.w. support/evolution. Is there a plan to address documentation shortcomings? How will you ensure quality, "after the fact," documentation?

3. Does the AMPS development process employ structured design/programming, inspections and walk-throughs, computer aided s.w. engineering, program design languages (PDL's)?

4. Do you employ a requirements matrix that shows how each system level requirement is satisfied by a particular module?

5. What method/procedure is being employed to control interface controls (e.g. both within, and outside the system)?

6. Does the philosophy "Thought first, regulation second," appropriately describe the AMPS development model?

7. Is the AMPS process accurately described by the following: Evolutionary development, maximum modularity, change-ability, and growth potential? If so, what are some examples?

8. What core group of metrics is/will be employed to track the AMPS process/progress within your office, within the PM's office? Why are/were these particular metrics chosen?

9. Of the metric set employed what is/will be the frequency in which they are monitored/reported?

10. Is there a plan for metrics evolution throughout the development process as changes in data needs, processing, and analysis evolve? Additionally, are adjustments/refinements made to the metric set in areas in which progress is good or level of data aggregation may be increased?

11. Is/will metric selection dependent on the CMM level of your office?

12. Does the PM have an in-house capability for metrics monitoring, or is an outside agent being employed in this function?

AMPS Development Process-Applied Management Principles (Cont'd)

13. Of the following list of metrics, which are available to your office/PM office on a monthly basis?:

- S.W size/cost status
- Manpower application status
- Cost/schedule status
- Defects/faults/errors/fixes
- Test program status
- Resource margins
- Quantitative s.w. spec. status
- Design/development status
- S.W. problem report status
- Delivery status

14. Is a cost model, such as COCOMO, used with the AMPS process?

15. Does your office use SLOC or FP's to measure size of the program?

16. In the area of manpower metrics, what is your office's ratio of total to experienced personnel? 6:1, 5:1, 4:1, 3:1, 2:1, 1:1?

17. Have manpower metrics been tailored to track the staffing for each : 1) Development task, 2) Skill (e.g. Ada, Database mngt. systems , 3) Organization (e.g. S.W., Q.A, Test, etc.)?

18. Within the realm of Cost/schedule status metrics, how do you ensure visibility into the s.w. development status? Is the s.w. WBS adequately defined? Please explain. For instance, is the WBS product-oriented in nature?

19. Is each CSCI tracked separately?

20. Are resource margin (e.g. CPU/Memory, I/O utilization) metrics being closely monitored both for the current host system and the target system?

21. For specifications metrics, how/when were the AMPS requirements baselined? For example, did this occur during the specification requirements review (SRR)? Additionally, did the AFMSS function as the initial AMPS baseline?

22. Can you cite a few ways in which the AMPS process focuses on defect prevention and early fault detection?

23. What is a "ballpark" figure for the AMPS s.w. problem report (SPR) rate/range? Typical rate is 5-30 SPR's/1000 SLOC.

24. Internal/external incremental delivery status metrics seem like a good idea for a program such as the AMPS; Is this being accomplished? Examples include tracking internal delivery to test organizations or external delivery to IV&V organizations.

AMPS Development Process-Risk Management

1. Risk management is paramount in the development of a program like the AMPS. A few procedures/techniques that the program seems to employ are: 1) Evolutionary design/delivery, 2) Rapid prototype/user involvement, 3) Use of COTS/NDI technology (e.g. AFMSS), 4) Metrics application, 5) Employment of IV&V (?) agent. Is this list accurate, and can it be added to? Please explain.

AMPS Development Process-Contract Considerations

1. What is the current contract arrangement for the AMPS program? For example, what is the type of contract employed, and if incentives are employed, how are they structured (e.g. what is incentivized and why)?

2. What alternative contract alternatives are being considered for the AMPS production model? For example, will the work continue to be conducted in-house, or will an outside contractor be utilized? What will be the contract type of choice, and how will any incentives be structured?

LIST OF REFERENCES

1. Attanasio, Henry, "*Contracting For Embedded Computer Software Within the Department of the Navy.*" Masters Thesis, Naval Postgraduate School, Monterey, June 1990.

2. Brykczynski, Bill, et al., "*Software Inspections: Eliminating Software Defects,*" briefing prepared by the Institute for Defense Analysis, February 5, 1993.

3. CECOM 93, Research, Development and Engineering Center, *Promotional Handbook*, 1993.

4. Charette, Robert N., So*ftware Engineering Risk Analysis and Management*, McGraw-Hill Book Co., New York, 1989.

5. Daskalantonakis, Michael J., Motorola, Inc. "*Achieving Higher SEI Levels,*" Crosstalk-Journal of Defense Software Engineering, September 1995.

6. Defense Systems Management College, *Mission Critical Computer Resources Management Guide*, 1989.

7. Department of Defense, MIL-STD-498, *System Software Development and Documentation*, 5 December 1995.

8. Department of the Air Force, Software Technology Support Center, "*Guidelines for Successful Acquisition of Software Intensive Systems,*" February 1995.

9. Development Team Questionnaire, Aviation Mission Planning System, (Appendix), August 1995.

10. GAO Report, "*Defense Doesn't Know What it Spends on Software,*" July 1992.

11. Guenther, Otto, LT. Gen., USA. "*An Army Perspective on Software Development.*" Crosstalk-The Journal of Defense Software Engineering, May 1995.

12. Jones, Capers, A*pplied Software Measurement,* McGraw-Hill Book Co., New York, 1991.

13. Jones, Capers, *Assessment and Control of Software Risks*, Yourdon Press, 1994.

14. Jones, Capers, *Programming Productivity*, McGraw-Hill Book Co., New York, 1986.

15. Kindl, Mark R., LTC, USA. "*Software Quality and Testing: What DoD can Learn from Commercial Practices,*" U.S. Army Institute for Research in Management Information Systems and Computer Sciences, Atlanta, August 1992.

16. Kitfield, James, "*Is Software DoD's Achilles' Heel?*" Article, Military Forum, July 1989.

17. Machiavelli, Niccolo, *The Art of War,* 1521.

18. Marciniak, John J., and Donald J. Reifer, Software Acquisition Management: Managing the Acquisition of Custom Software Systems, John Wiley and Sons, New York, 1990.

19. Marshall, A.J., PROSOFT, Inc., "*De-mystifying Software Configuration Management,*" Crosstalk- The Journal of Defense Software Engineering, May 1995.

20. Mullins, Thomas, E. "*Impact of Adopting Commercial Practices In Software Development and Maintenance.*" Masters Thesis, Naval Postgraduate School, Monterey, March 1995.

21. Paulk, Mark C., Bill Curtis, Mary Beth Chrissis, and Charles V. Weber, *Capability Maturity Model for Software Version 1.1,* Software Engineering Institute, CMU/SEI-93-TR-24, February 1993.

22. Perry, William J., Secretary of Defense, Specifications and Standards -- *A New Way of Doing Business,* Memorandum, 29 June 1994.

23. Program Manager, Avionics, PEO Aviation, *Configuration Management Process Description and Diagram,* September 1993.

24. Program Manager, Avionics, PEO Aviation, *Functional Description Document (Operational Concept),* Aviation Mission Planning System, February 1995.

25. Program Manager, Avionics, PEO Aviation, *Program Review Briefing Packet,* Aviation Mission Planning System Working Group, August 1995.

26. Program Manager Avionics, PEO Aviation, *Users Guide, Aviation Mission Planning System,* March 1995.

27. Project Manager, Aviation Electronic Combat "*Acquisition Strategy Decision Paper,*" Aviation Mission Planning System, February, 1994.

28. Project Manager, Aviation Electronic Combat, *Test and Evaluation Master Plan, U. S. Army Maneuver Control System, Army Aviation Planning System (AMPS)*, July 1995.

29. Project Manager, Aviation Electronic Combat, *Top Level System Requirements Document*, Aviation Mission Planning System, June 1995.

30. Project Office, Aviation Mission Planning System (C2SID), *AMPS Capabilities Matrix*, August 1995.

31. Project Office, Aviation Mission Planning System (C2SID), Interviews, 30 August 1995.

32. Project Office, Aviation Mission Planning System (C2SID), Interviews, 31 August 1995.

33. Radatz, Olson, Campbell, Logicon, Inc., "*MIL-STD-498*," Crosstalk-Journal of Defense Software Engineering, February 1995.

34. Shimeall, Timothy J., "*Software Engineering Developments*," Technology Review and Update Seminar for Technical Personnel, Naval Postgraduate School, Monterey, April 1995.

35. Shimeall, Timothy J., *Technology Review and Update Seminar for Technical Personnel*, Naval Postgraduate School, Monterey, April 1995.

36. Software Engineering Directorate (SED-AV), CECOM, *Activities in Support of AMPS Development and PDSS* (Briefing Packet), August 1995.

LIST OF INTERVIEWS

1. Andreolo, James, Software Engineering Directorate - Avionics (SED-AV), CECOM-RDEC, Ft. Monmouth, NJ, August 1995.

2. Banks, Hayes, Vitronics, Inc., AMPS Project Office (C2SID), CECOM, RDEC, Ft. Monmouth, NJ, August 1995.

3. Bahary, John, Software Development Chief, C2SID, CECOM- RDEC, Ft. Monmouth, NJ, August 1995.

4. Carpenter, Cindy, Configuration Manager, C2SID, CECOM- RDEC, Ft. Monmouth, NJ, August 1995.

5. Donnely, John, System Dynamics International (SDI), PM-AEC, PEO AVN, St. Louis, MO., August 1995.

6. Malinowski, Robert, Software Engineering Directorate-Avionics (SED-AV), CECOM-RDEC, Ft. Monmouth, NJ, August 1995.

7. Sova, Robin, System Dynamics International (SDI), PM-AEC, PEO AVN, St. Louis, MO., August 1995.

8. Tang, Dzung, Software Engineering Directorate - Avionics (SED-AV), CECOM-RDEC, Ft. Monmouth, NJ, August 1995.

9. Tom, Anthony, AMPS Project Leader, C2SID, CECOM-RDEC, Ft. Monmouth, NJ, August 1995.

10. Williams, Lennox, Software Engineering Directorate- Avionics (SED-AV), CECOM-RDEC, Ft. Monmouth, NJ, August 1995.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center... 2
   8725 John J. Kingham Road, STE 0944
   Fort Belvoir, VA 22060-6218

2. Library, Code 13.. 2
   Naval Postgraduate School
   Monterey, CA 93943-5101

3. Defense Logistics Studies Information Exchange... 1
   U.S. Army Logistics Management College
   Fort Lee, Virginia 238016043

4. Acquisition Library... 1
   Department of Systems Management
   Naval Postgraduate School
   Monterey, CA 93943-5103

5. OASA (RDA).. 1
   ATTN: SARD-ZAC
   103 Army Pentagon
   Washington, DC 20310

6. Professor David V. Lamm (Code SM/Lt) 5
   Naval Postgraduate School
   Monterey, California 93943-5103

7. Professor Martin J. McCaffrey (Code SM/MF).. 6
   Naval Postgraduate School
   Monterey, California 93943-5100

8. Professor Orin E. Marvel (Code CC/OM).. 2
   Naval Postgraduate School
   Monterey, California 93943-5100

9. LTC John T. Dillard (Code SM/Dj)... 1
   Naval Postgraduate School
   Monterey, California 93943-5100